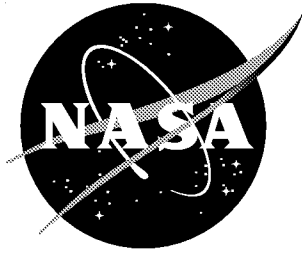


NASA/TM-2001-211049



Buckets: Smart Objects for Digital Libraries

*Michael L. Nelson
Langley Research Center, Hampton, Virginia*

August 2001

The NASA STI Program Office ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

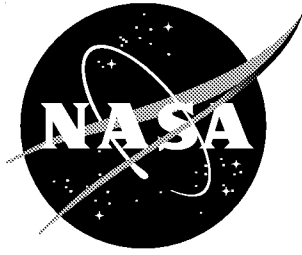
- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.
- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.
- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results ... even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at <http://www.sti.nasa.gov>
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA STI Help Desk at (301) 621-0134
- Phone the NASA STI Help Desk at (301) 621-0390
- Write to:
NASA STI Help Desk
NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320

NASA/TM-2001-211049



Buckets: Smart Objects for Digital Libraries

*Michael L. Nelson
Langley Research Center, Hampton, Virginia*

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

August 2001

Available from:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

ABSTRACT

BUCKETS: SMART OBJECTS FOR DIGITAL LIBRARIES

Michael L. Nelson
Old Dominion University, 2000
Director: Dr. Kurt Maly

Current discussion of digital libraries (DLs) is often dominated by the merits of the respective storage, search and retrieval functionality of archives, repositories, search engines, search interfaces and database systems. While these technologies are necessary for information management, the information content is more important than the systems used for its storage and retrieval. Digital information should have the same long-term survivability prospects as traditional hardcopy information and should be protected to the extent possible from evolving search engine technologies and vendor vagaries in database management systems. Information content and information retrieval systems should progress on independent paths and make limited assumptions about the status or capabilities of the other.

Digital information can achieve independence from archives and DL systems through the use of buckets. Buckets are an aggregative, intelligent construct for publishing in DLs. Buckets allow the decoupling of information content from information storage and retrieval. Buckets exist within the Smart Objects and Dumb Archives model for DLs in that many of the functionalities and responsibilities traditionally associated with archives are “pushed down” (making the archives “dumber”) into the buckets (making them “smarter”). Some of the responsibilities imbued to buckets are the enforcement of their terms and conditions, and maintenance and display of their contents. These additional responsibilities come at the cost of storage overhead and increased complexity for the archived objects. However, tools have been developed to manage the complexity,

and storage is cheap and getting cheaper; the potential benefits buckets offer DL applications appear to outweigh their costs.

We describe the motivation, design and implementation of buckets, as well as our experiences deploying buckets in two experimental DLs. We also introduce two modified forms of buckets: a “dumb archive” (DA) and the Bucket Communication Space (BCS). DA is a slightly modified bucket that performs simple set management functions. The BCS provides a well-known location for buckets to gain access to centralized bucket services, such as similarity matching, messaging and metadata conversion. We also discuss experiences learned from using buckets in the NCSTRL+ and Universal Pre-print Server (UPS) experimental digital libraries. We conclude with comparisons to related work and discussion about possible areas for future work involving buckets.

ACKNOWLEDGMENTS

This dissertation was made possible through the assistance, encouragement and patience of many people. Foremost among these people are the members of my committee. Kurt Maly provided the direct advisement, insight and strategic vision necessary for the definition, refinement and wide adoption of the research results. Stewart Shen and Mohammad Zubair were constant supporters and more than occasionally devil's advocates during our weekly meetings. Frank Thames provided much of my initial motivation to pursue a Ph.D., and David Keyes' encouragement is the reason I chose to obtain it at Old Dominion University.

Many fellow students at Old Dominion University have positively affected my research. Xiaoming Liu, Mohamed Kholief, Shanmuganand Naidu, Ajoy Ranga, and Hesham Anan are among those that have made design or coding suggestions, developed supporting technologies, and ferreted out many bugs.

NASA Langley Research Center has provided me with the opportunity and resources to perform digital library research and development. These current and former NASA colleagues have provided technical, financial and moral support in the breadth of my digital library activities at Langley: David Bianco, Aileen Biser, David Cordner, Delwin Croom, Sandra Esler, Gretchen Gottlich, Nancy Kaplan, Mike Little, Ming-Hokng Maa, Mary McCaskill, Daniel Page, Steve Robbins, Joanne Rocker, George Roncaglia, and Melissa Tiffany.

A number of people outside Old Dominion University and Langley Research Center played significant roles in supporting the development and adoption of buckets. Among these people are: Herbert Van de Sompel (University of Ghent), Marcia Dreier (Air Force Research Laboratory) and Rick Luce (Los Alamos National Laboratory).

Finally, I would like to thank Rod Waid for the creation of the lovable "Phil" character that eventually evolved into our research group's mascot, and Danette Allen for her patience and support.

TABLE OF CONTENTS

	PAGE
LIST OF TABLES	vii
LIST OF FIGURES	viii
 Chapter	
1. INTRODUCTION.....	1
2. MOTIVATION AND OBJECTIVES.....	6
2.1 Why Digital Libraries?	6
2.1.1 Digital Libraries vs. the World Wide Web	8
2.1.2 Digital Libraries vs. Relational Database Management Systems	9
2.2 Trends in Scientific and Technical Information Exchange.....	10
2.3 Information Survivability	13
2.4 Objectives and Design Goals	15
2.4.1 Aggregation	15
2.4.2 Intelligence	16
2.4.3 Self-Sufficiency	16
2.4.4 Mobility	17
2.4.5 Heterogeneity.	18
2.4.6 Archive Independence	18
3. BUCKET ARCHITECTURE	19
3.1 Overview	19
3.2 Implementation	24
3.2.1 Bucket Methods	26
3.2.2 File Structure	30
3.2.3 Terms and Conditions	33
3.2.4 Internal Bucket Operation.....	38
3.2.5 Metadata Extensions	39
3.3 Discussion	41
3.3.1 Bucket Preferences	41
3.3.2 Systems Issues	43
4. DUMB ARCHIVES	47
4.1 Overview	47
4.1.1 The SODA DL Model	47
4.1.2 Archive Design Space	49
4.1.3 Publishing in the SODA Model	50
4.2 Implementation	51

	PAGE
4.2.1 Implemented Methods	52
4.2.2 Changes from a Regular Bucket	52
4.3 Discussion	53
4.3.1 DA Examples	53
4.3.2 DBM Implementation Notes	56
4.3.3 Open Archives Initiative Dienst Subset Mapping	57
5. BUCKET COMMUNICATION SPACE	60
5.1 Overview	60
5.1.1 File Format Conversion	61
5.1.2 Metadata Conversion	61
5.1.3 Bucket Messaging	62
5.1.4 Bucket Matching	62
5.2 Implementation	63
5.2.1 Implemented Methods	63
5.2.2 Changes from a Regular Bucket	68
5.3 Discussion	69
5.3.1 Performance Considerations	69
5.3.2 Current Limitations	72
6. BUCKET TESTBEDS	74
6.1 NCSTRL+	74
6.1.1 Dienst	74
6.1.2 Clusters	76
6.2 Universal Preprint Server	77
6.2.1 Lightweight Buckets	78
6.2.2 SFX Reference Linking in Buckets	79
7. RELATED WORK	83
7.1 Aggregation	83
7.1.1 Kahn/Wilensky Framework and Derivatives	83
7.1.2 Multivalent Documents	83
7.1.3 Open Doc and OLE	84
7.1.4 Metaphoria	84
7.1.5 VERS Encapsulated Objects	84
7.1.6 Aurora	85
7.1.7 Electronic Commerce	85
7.1.8 Filesystems and File Formats	86
7.2 Intelligence	86
7.3 Archives	87

	PAGE
7.4 Bucket Tools	88
8. FUTURE WORK	92
8.1 Alternate Implementations	92
8.1.1 Buckets	92
8.1.2 Dumb Archives	93
8.1.3 Bucket Communication Space	93
8.2 Extended Functionality	93
8.2.1 Pre-defined Packages and Elements	94
8.2.2 XML Metadata	94
8.2.3 More Intelligence	95
8.3 Security, Authentication and Terms & Conditions	95
8.4 New Applications	97
8.4.1 Discipline-Specific Buckets	97
8.4.2 Usage Analysis	97
8.4.3 Software Reuse	98
9. RESULTS AND CONCLUSIONS	99
REFERENCES	102
APPENDICES	
A. BUCKET VERSION HISTORY	114
B. BUCKET API	118
C. DA API	150
D. BCS API	156

LIST OF TABLES

TABLE	PAGE
1. System configurations used for bucket testing.....	25
2. Bucket API	27
3. Reserved packages	32
4. Directives available in T&C files	37
5. Bucket preferences	42
6. The archive design space	50
7. DA API	52
8. OAi → DA mapping	59
9. BCS API	64
10. UPS participants	78

LIST OF FIGURES

FIGURE	PAGE
1. DL component technologies	9
2. Pyramid of publications for a single project/concept	11
3. Pyramid of publications rests on unpublished STI	12
4. STI lost over time	15
5. Model of a typical NASA STI bucket	21
6. Sample project bucket	22
7. Sample course bucket	23
8. Output of the “display” method	28
9. Thumbnails in the “display” method	31
10. Bucket structure	32
11. RFC-1807 metadata	40
12. The three strata of DLs	48
13. The SODA publishing model	51
14. Population of the DA	54
15. DA query (?method=da_put&adate=<20000101)	55
16. DA query (?method=da_put&adate=19940101-20000101&subject=cs)	55
17. DA query (?method=da_put&subject=phys)	56
18. NACA bucket before similarity matching	66
19. NACA bucket after similarity matching	67
20. Sample similarity matching matrix	70
21. Partitioning of the similarity matching matrix	72
22. NCSTRL+ lineage	74
23. A UPS bucket with SFX buttons	80
24. SFX interface	81
25. Creation tool	89

FIGURE	PAGE
26. Management tool	90
27. Administration tool	91

CHAPTER ONE

INTRODUCTION

Although digital libraries (DLs) pre-date the World Wide Web (WWW) (Berners-Lee, Cailliau, Groff, & Pollermann, 1992), the popularity and prevalence of the WWW has focused attention on DLs for both the general user and research communities. The WWW provides ubiquitous access to distributed information content. However, finding information in the WWW can be difficult. It is estimated that the best WWW search engines contain less than 35% of the total indexable WWW, with some as little as 3% (Lawrence & Giles, 1998). DLs are seen as a way to define gardens of information in a vast, untamed forest of spurious information resources. DLs are now commonly used in science, technology, arts and humanities. In some cases, they provide an on-line analogue of traditional libraries, but without the geographic or temporal limitations. In other cases, DLs are being used to create and disseminate collections of information that had not been previously feasible or possible to collect in traditional libraries.

We begin with the observation that information content is more important than the systems used to store and retrieve it. While this seems obvious enough, this fact is often obscured during discussions of DLs. Instead, the focus of DL discussions is primarily on the merits of specific relational database managers (RDBMs), search engines, the programming language or systems used, and other implementation specific details. This is because when a specific DL implementation is chosen, the services it provides (e.g., searching, browsing, document access) are often vertically integrated with the content it services, sometimes done purposefully, in an attempt to control the intellectual property rights to the object. However, such tight integration is at odds with the goals of easily transitioning to future DL systems and concurrent support of multiple DL access to a single collection of data objects. Even in many DL systems that have the direct goal

of having an open architecture, with multiple searching, browsing and other user interfaces possible, there is an assumption of tightly tying the data objects to a single service that controls their access. For example, in the open architecture DL proposal of Lagoze & Payette (1998), the integration of repository and object is explicitly stated:

“The repository service provides the mechanism for the deposit, storage, and access to digital objects. A digital object is considered contained within a repository if the URN of that object resolves to the respective repository (and, thus, access to the object is only available via a service request to that repository).”

Our approach begins with promoting the importance of the information objects above that of the DL systems used for their storage, discovery, and management. Within the context of DLs, we make the information objects “first-class citizens”. We propose decoupling information objects from the systems used for their storage and retrieval, allowing the technology for both DLs and information content to progress independently. Paepcke (1996) argues that “searching is not enough” and that DLs need to provide a wide range of value-added services, far more than DLs currently provide. We agree with this position, and feel that dismantling the current stovepipe of “DL-archive-content” is the first step in building richer DL experiences for users.

To demonstrate this partitioning between DLs, archives and information content, we introduce “buckets”. Buckets are aggregative, intelligent, object-oriented constructs for publishing in digital libraries. They are partially similar in design to Kahn-Wilensky Digital Objects (DOs) (Kahn & Wilensky, 1995), but with a few significant differences and are optimized for DL applications. Although buckets could accurately be described as “archivelets”, the name “buckets” was chosen for several reasons: First of all it is easy to pronounce and has a strong visual metaphor for its aggregation capability. Most importantly, the target user community (not all of which are computer scientists) warmed to it more than variations on “object”, “package” and other popular computer science terms.

Buckets exist within the “Smart Objects, Dumb Archives” (SODA) DL model (Maly, Nelson, & Zubair, 1999). The SODA DL model dictates that functionalities traditionally associated with archives are pushed down into the buckets, making the buckets “smarter” and the archives “dumber”. Some of a bucket’s responsibilities include: storing, tracking, and enforcing its own terms and conditions (T&C); maintenance, display and dissemination of its contents; maintaining its own logs of actions and errors; and informing appropriate parties when certain events occur. Buckets provide mechanism, not policy. Buckets have no assumptions about their content, T&C, their deployment profile or other matters. However, the mechanisms that buckets and their related tools provide should be sufficient to implement an organization’s policy.

The motivation for buckets came from previous experience in the design, implementation and maintenance of NASA scientific and technical information (STI) DLs, including the Langley Technical Report Server (LTRS) (Nelson, Gottlich, & Bianco, 1995; Nelson & Gottlich, 1994), the NASA Technical Report Server (NTRS) (Nelson, Gottlich, Bianco, et al., 1995), and the NACA Technical Report Server (NACATRS) (Nelson, 1999). Buckets can trace their evolution back to the NACATRS project, which featured what we now call “proto-buckets”. Objects in the NACATRS had many of aggregation features of buckets, but lacked the additional features such as intelligence and did not have a well-defined application programming interface (API).

In early user evaluation studies on these DLs, one reoccurring theme was detected. While access to the technical report (or re/pre-print) was desirable, users particularly wanted access to the raw data collected during the experiments, the software used to reduce the data, and the ancillary information that went into the production of the published report (Roper, McCaskill, Holland, et al., 1994). The need for NASA research projects to deliver not just a report, but also software and supporting technologies was identified as early as 1980 (Sobieski, 1994), but NASA’s treatment of non-report STI has remained uneven. Reports continue to receive the primary focus, and the interest and capacity to archive and disseminate other information types (data, notes, software, audio,

video) ebbs and flows. The interest here is to create a set of capabilities to permit DLs to accommodate requests for substantially more information than just finalized reports. However, rather than setup separate DLs for each information type or stretch the definition of a traditional report to include various multi-media formats, the desire was to define an arbitrary digital object that could capture and preserve the potentially intricate relationship between multiple information types.

Additionally, our experiences with updating the DLs and making the content accessible through other DLs and web-crawlers led to the decision to make the information objects intelligent. We wanted the objects to receive maximum exposure, so we did not want them “trapped” inside our DLs, with the only method for their discovery coming from our DL interface. However, the DL should have more than just an exportable description of how to access the objects in the DL. The information object should be independent of the DL, with the capability to exist outside of the DL and move in and out of different DLs in the future. However, to not assume which DL was used to discover and access the buckets means that the buckets must be self-sufficient and perform whatever tasks are required of them, potentially without the benefit of being arrived at through a specific DL. Multiple implementations of buckets are possible. However, for the bucket implementation presented here, the following requirements must be met for the computer hosting the buckets:

- a hypertext transfer protocol (http) (Fielding, Gettys, Mogul, et al., 1999) server that implements the common gateway interface (CGI) specification.
- a Perl 5 interpreter (Wall, Christiansen, & Schwarz, 1996) that the bucket can find.

As long as these two requirements are met, the buckets will be able to function. The buckets have a “bunker” mentality: even if the various search engines, DLs and other resources normally used for their discovery moves, breaks, or otherwise degenerates,

buckets should continue to function. The well being of a bucket depends on the lowest possible common denominator: a CGI http server and Perl interpreter, and not on more complex and possibly transient DL services.

The outline for the rest of this thesis is as follows: Chapter Two provides the motivation for DLs and buckets, and design goals of buckets. Chapter Three discusses the bucket architecture and implementation. Chapter Four discusses the dumb archive architecture and implementation. Chapter Five discusses the architecture and implementation of the Bucket Communication Space. Chapter Six describes how buckets were used in two prototype DLs: NCSTR+ and the Universal Preprint Service (UPS). Chapter Seven compares and contrasts buckets with related work, and Chapter Eight discusses some of the possible future work. Chapter Nine provides the conclusions and summary.

CHAPTER TWO

MOTIVATION AND OBJECTIVES

2.1 Why Digital Libraries?

The preservation and sharing of its intellectual output and research experiences is the primary concern for all research institutions. However, in practice information preservation is often difficult, expensive and not considered during the information production phase. For example, Henderson (1999) provides data showing for the period of 1960-1995 that “knowledge conservation grew half as much as knowledge output”, as a result of research library funding decreasing relative to increasing research and development spending (and a corresponding increase in publications). In short, more information is being produced, and it is being archived and preserved in fewer libraries, with each library having fewer resources. Though eloquent arguments can be presented for the role for and purpose of traditional libraries and data can be presented for the monetary savings libraries can provide (Griffiths & King, 1993), the fact remains that traditional libraries are expensive. Furthermore, the traditional media formats (i.e. paper, magnetic tapes) housed in the traditional libraries are frail, requiring frequent upkeep and are subject to environmental dangers (Lesk, 1997; United States General Accounting Office, 1990). DL technologies have allowed some commercial publishers to become more involved with library functions, serving on the WWW the byproducts of their publishing process (PostScript, PDF, etc.). However, ultimately the goals of publishers and the goals of libraries are not the same, and the long-term commitment of publishers to provide library-quality archival and dissemination services is in doubt (Arms, 1999). While not a panacea, an institution’s application of DL technologies will be an integral part of their knowledge usage and preservation effort, in either supplanting or supplementing traditional libraries.

All of this has tremendous impact on a U.S. Government agency like NASA. Beyond attention grabbing headlines for its various space programs, NASA ultimately produces information. The deliverables of NASA's aeronautical and space projects are information for either a targeted set of customers (e.g., Boeing) or for science and posterity. The information deliverables can have many forms: publications in the open literature; a self-published technical report series; and non-traditional STI media types such as data and software. NASA contributions to the open literature are subject to the same widening gap in conservation and output identified by Henderson (1999). For some, the NASA report series is either unknown or hard to obtain (Roper, McCaskill, Holland, et al., 1994). For science data, NASA has previously been criticized for poor preservation of this data (United States General Accounting Office, 1990). However, NASA has identified and is addressing these problems with ambitious goals. From the NASA STI Program Plan (NASA, 1998):

“By the year 2000, NASA will capture and disseminate all NASA STI and provide access to more worldwide mission-related information for its customers. When possible and economical, this information will be provided directly to the desktop in full-text format and will include printed material, electronic documentation, video, audio, multimedia products, photography, work-in-progress, lessons-learned data, research laboratory files, wind tunnel data, metadata, and other information from the scientific and technical communities that will help ensure the competitiveness of U.S. aerospace companies and educational institutions.”

Although tempered with the phrase “possible and economical”, it is clear that the expectations are much higher than simply automating traditional library practices. Much of the STI identified above has historically not been included in traditional library efforts, primarily because of the mismatch in hard- and soft-copy media formats. However, the ability to now document the entire research process and not just the final results presents entirely new challenges about how to acquire and manage this increased volume of information. To effectively implement the above mandate, additional DL technology is required.

2.1.1 Digital Libraries vs. the World Wide Web

A common question regarding DLs is “Why not just use existing WWW tools/methods?” Indeed, most DLs use the WWW as the access and transport mechanism. However, it is important to note that while the WWW meets the rapidity requirement of STI dissemination, it has no intrinsic management or archival functions. Just as a random collection of books and serials do not make a traditional library, a random collection of WWW pages does not make a DL. A DL must possess acquisition, management, and maintenance processes. These processes will vary depending on the customers, providers and nature of the DL, but these processes will exist in some format, implicitly or explicitly.

There have been proposals to subvert the traditional publication process with authors self-publishing from their own WWW pages (Harnad, 1997). However, while this availability is useful, pre-prints (or re-prints) linked from a researcher’s personal home page are less resilient to changes in computer infrastructure, organization changes, and personnel turnover. Ignoring the socio-political issues of (digital) collegial distribution, there is an archival, or longevity, element to DLs which normal WWW usage does not satisfy. The average lifetime of a uniform resource locator (URL) has been estimated at 44 days (Kahle, 1997), clearly insufficient for traditional archival expectations. Uniform Resource Names (URNs) can be used to address the transient nature of URLs. URNs provide a unique name for a WWW object that can be mapped to a URL by a URN server. The relationship between URNs and URLs is the same as Internet Protocol (IP) names and IP addresses, respectively. CNRI Handles (Sun & Lannom, 2000), Persistent URLs (Purls) (Shafer, Weibel, Jul, & Fausey, 1996) and Digital Object Identifiers (DOIs) (Paskin, 1999) are some common URN implementations. However, no URN implementation has achieved the ubiquity of URL use, and significant maintenance is required to keep a large collection of URNs current. In summary, a DL defines a well-known location for STI to be placed, managed, and

accessed. Given the prevalence of the WWW, the well-known location that a DL provides is likely to be WWW accessible.

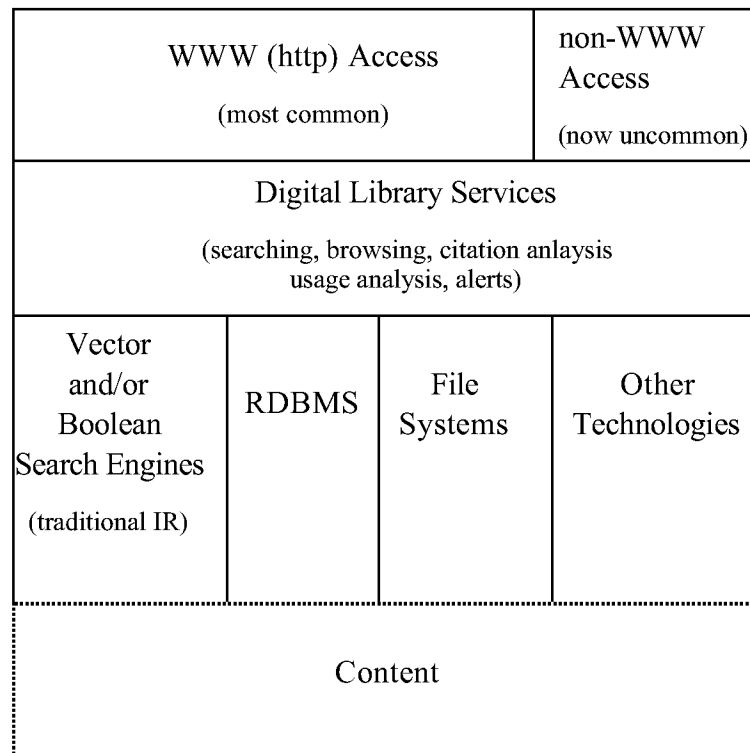


FIG. 1. DL component technologies.

2.1.2 Digital Libraries vs. Relational Database Management Systems

Perhaps the second most common question after “Why not just use the WWW?” is “Why not just use a database?” The answer to the database question is subtler. Generally relational databases are less well suited for more generalized information retrieval (IR) requirements typical of library applications, which often feature boolean or vector search engines. Two main differences between traditional IR systems and relational databases management systems (RDBMS) is that the data objects in IR systems are documents, which are less structured than the tables of relations which are the data objects in RDBMs (Frakes & Baeza-Yates, 1992). Also, retrieval in an IR system is probabilistic, as opposed to deterministic in a RDBMS (Frakes & Baeza-Yates, 1992). Some commercial and professional society DLs are constructed with web pages indexed

by traditional IR search engines, including the Institute for Electrical and Electronics Engineers (IEEE) Computer Society DL, which uses the “Autonomy” search engine and D-Lib Magazine DL which uses the “Excite” search engine.

However, it is possible to use a RDBMS to build a DL, especially if high-quality, structured metadata is available from which tables can be built. This is the approach of the IBM “DB2 DL” commercial product and the Association for Computing Machinery (ACM), which uses an “Oracle” RDBMS for its DL. A DL is the union of its content and the services it provides on that content. A traditional IR search engine or a RDBMS, insofar as they provide only a single service (searching), are just components of a DL, not the DL itself. The relationship between the WWW, traditional IR search engines, DLs, RDBMS, and other technologies is illustrated in Figure 1.

2.2 Trends in Scientific and Technical Information Exchange

Rapidity and breadth of communication have always been significant requirements in the exchange of STI. Scientific journals evolved in the 17th century to replace the system of exchanging personal letters between scientists, which evolved because of unacceptable delays in publishing books (Odlyzko, 1995). However, journals are no longer used for rapid communication, but rather as “a medium for priority claiming, quality control and archiving scientific work.” (Bennion, 1994). To achieve rapid communication of STI, different disciplines have adopted various models. Starting in the 1960’s, “Letters” journals began to appear in some disciplines to offer more rapid dissemination of research results, while in other disciplines the pre-print or technical report emerged as the rapid dissemination vehicle (Vickery, 1999). In computer science, the technical report is a common unit of exchange. In disciplines such as high-energy physics, the pre-print culture is well established. Paul Ginsparg, a physicist active in digital libraries, notes that “The small amount of filtering provided by refereed journals plays no effective role in our research.” (Ginsparg, 1994). While noting that not all disciplines embrace the pre-print / technical report culture equally, Odlyzko (1995) states “it is rare for experts in any mathematical subject to learn of a major new development in

their area through a journal publication” and also relates comments by computer scientists Rob Pike (“that in his area journals have become irrelevant”) and Joan Feigenbaum (“if it didn’t happen at a conference, it didn’t happen”).

A journal article is often only a fraction of the available technical literature about a given subject. Theses, dissertations, conference papers, and technical reports are known as “grey literature” and receive varying degrees of peer review. “White literature,” available through standard publications channels and processes, is often supported by a larger body of grey literature. The role of the large amount of grey literature and its relation to the smaller amount of white literature, and the issues associated with integrating the two have been present since the post-World War II U.S. Government sponsored research boom (Bennington, 1952; Gray, 1953; Scott, 1953). David Patterson, co-inventor of the RISC computer chip, noted that in one of his first research projects, the output was 2 journal articles, 12 conference papers, and 20 technical reports (Patterson, 1994). If we consider this pyramid of publications (Fig. 2) to be typical, then a journal article actually functions as an abstract of a larger body of STI.

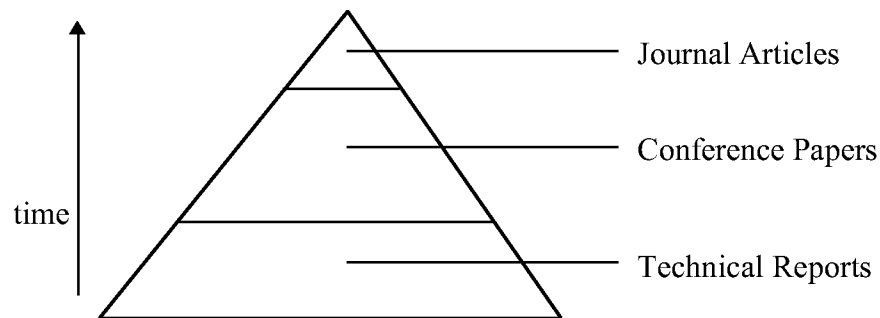


FIG. 2. Pyramid of publications for a single project/concept.

It is estimated that there are over 100,000 domestic technical reports produced annually (Esler & Nelson, 1998). The result is that even if there are 20,000 primary research journals (Bennion, 1994), they do not represent the entirety of STI. These numbers do not include 1) confidential, secret, proprietary, and otherwise restricted reports; or 2) non-report STI, such as computer software, data sets, video, geographic

data, etc. Indeed, anecdotal evidence suggests that the WWW is not just a rapid transport mechanism for white and grey literature, but collections of WWW pages are becoming a new unit of STI exchange as well. Figure 3 shows the Pyramid of Publications described in Figure 2 resting on a larger body of unpublished STI.

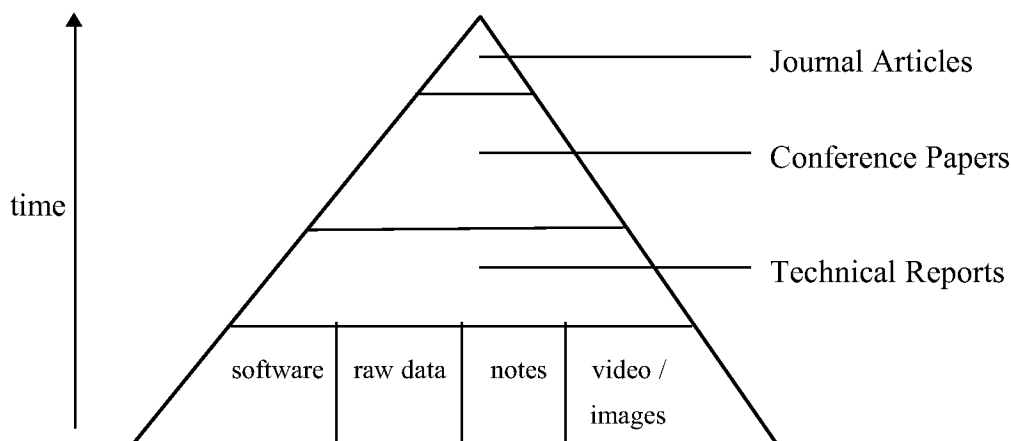


FIG. 3. Pyramid of publications rests on unpublished STI.

Schatz and Chen (1996) give a summary of the Digital Library Initiative (DLI) projects focusing on building large digital libraries of non-report STI. However, these efforts can be summarized as propagating a “separate but equal” philosophy with regards to non-report STI. Instead of integrating software, datasets, etc. into the same DL, which contains the reports, separate DLs are created for the new collection. The researcher is still left to reconstruct the original information tuple by integrating search results from various DLs. The DLI2 initiative (Lesk, 1999; Griffin, 1999), a follow-on to the 1994-1998 DLI, is funding a broader range of DL projects, including a great number with focus on non-report literature. However, these projects still do not focus on redefining the output of the STI research process. We consider “separate-but-equal” DLs to be harmful. For example, no matter how sophisticated a video DL becomes, the video should never be de-integrated from the data sets that supplement the video, the software used to process the data sets, and the report that documents the entire project. The limitations of current STI exchange mechanisms can be summarized as follows:

- *highly focused on journal articles*, despite their decreasing value to researchers and practitioners in some fields;
- *inadequate acquisition of grey literature*, the grist of technical exchange; and
- *inability to integrate non-publication media*, such as datasets, software, and video.

These limitations are largely side effects of the hard copy distribution paradigm. As STI exchange moves toward electronic distribution, existing mechanisms should not merely be automated, but the entire process should be revisited.

2.3 Information Survivability

The longevity of digital information is a concern that may not be obvious at first glance. While digital information has many advantages over traditional printed media, such as ease of duplication, transmission and storage, digital information suffers unique longevity concerns that hard copy does not, including short life spans of digital media (and their reading devices) and the fluid nature of digital file formats (Rothenberg, 1995; Lesk, 1997). The Task Force on Archiving of Digital Information (1996) distinguished between: *refreshing*, periodically copying the digital information to a new physical media; and *migrating*, updating the information to be compatible with a new hardware/software combination. Refreshing and migrating can be complex issues. The nature of refreshing necessitates a hardware-oriented approach (perhaps with secondary software assistance). Software objects cannot directly address issues such as the lifespan of digital media or availability of hardware systems to interpret and access digital media, but they can implement a migration strategy in the struggle against changing file formats. An aggregative software object could allow for the long-term accumulation of converted file formats. Rather than successive (and possibly lossy) conversion of:

Format1 → Format2 → Format3 → → FormatN

We should have the option of:

Format1 → Format2

Format1 → Format3

Format1 →

Format1 → FormatN

With each intermediate format stored in the same location. This would allow us to implement the “throw away nothing” philosophy, without burdening the DL directly with increasing numbers of formats.

For example, a typical research project at NASA Langley Research Center produces information tuples: raw data, reduced data, manuscripts, notes, software, images, video, etc. Normally, only the report part of this information tuple is officially published and tracked. The report might reference on-line resources, or even include a CD-ROM, but these items are likely to be lost, degrade, or become obsolete over time. Some portions such as software, can go into separate archives (i.e., COSMIC – the official NASA software repository) but this leaves the researcher to locate the various archives, then re-integrate the information tuple by selecting pieces from the different, and perhaps, incompatible archives. Most often, the software and other items, such as datasets are simply discarded or effectively lost in informal, short-lived personal archives. After 10 years, the manuscript is almost surely the only surviving artifact of the information tuple. The fate typical of various STI data types is depicted in Figure 4.

As an illustration, COSMIC ceased operation in July 1998; its operations were turned over to NASA’s technology transfer centers. However, at the time of this writing there appears to be no operational successor to COSMIC. Unlike their report counterparts in traditional libraries or even DLs such as LTRS, the software contents of COSMIC have been unavailable for several years, if not completely lost.

Additional steps can be taken to insure the survivability of the information object. Data files could be bundled with the application software used to process them, or if

common enough, different versions of the application software, with detailed instructions about the hardware system required to run them, could be a part of the DL. Furthermore, they could include enough information to guide the future user in selecting (or developing) the correct hardware emulator.

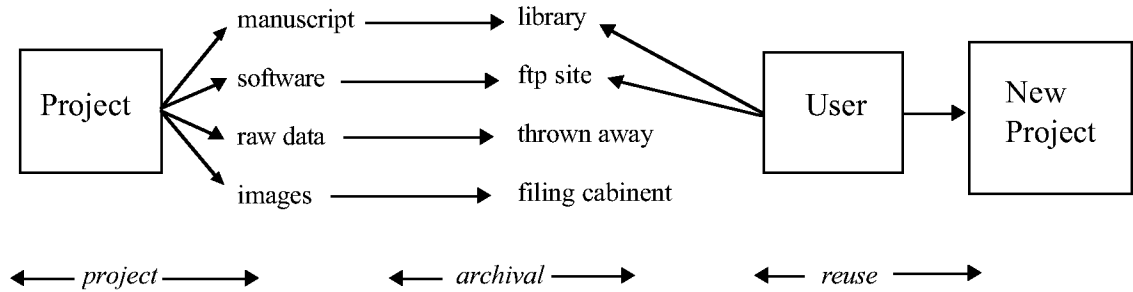


FIG. 4. STI lost over time.

2.4 Objectives and Design Goals

The objectives of this research are as follows:

1. Develop “buckets” – a collection of mechanisms and protocols to aggregate, protect, manage, and mobilize content and basic services.
2. Develop a reference implementation of buckets based on http, CGI and Perl.
3. Evaluate the concept and reference implementation in different application domains.

The development of buckets is guided by a number of design goals. As suggested by the SODA model, buckets have unique requirements due to their emphasis on minimizing dependence on specific DL implementations. The design goals are: aggregation, intelligence, self-sufficiency, mobility, heterogeneity and archive independence.

2.4.1 Aggregation

As outlined in the discussion given above, DLs should be shielded from the transient nature of data file formats and the information object should be allowed to evolve independently of the system(s) users employ to discover the information object.

Furthermore, a trend was noted that DLs were often built around the now obsolete media boundaries of traditional libraries (Esler & Nelson, 1998): technical reports existed in a technical report DL, images existed in an image DL, software existed in a software DL. A thesis of this study is that since all of these objects were created at the same time and potentially have subtle relationships between them, they should be able to be stored in the same information object. In NASA DLs there was an information hemorrhaging: a suite of information objects would be prepared digitally, but since the DLs could only accept a single information object (report or re-print), the other objects were left to be archived and distributed informally, if at all.

With decreasing costs of physical storage media, the cost of not saving data sets becomes more expensive than saving them. Some data, such as time-dependent satellite data or pilot-in-the-loop flight simulations, cannot be replaced or recreated. Buckets provide a way to aggregate all the related information objects, which could be useful for future, possibly unknown uses into a single container object that provides ease of maintenance. Buckets can also be used to aggregate the successive migrations of an information object from one hardware/software system to the next.

2.4.2 Intelligence

Yet another design goal of buckets is that they be autonomous and active, not passive and tied to a server. Buckets should be able to perform and respond to actions on their own and have them be active participants in their own state and existence. Buckets do not necessarily have to reveal their intelligence in interaction with users, but rather in interaction with tools and other buckets. Making information objects intelligent opens the door for whole new realms of applications. Although some bucket applications are obvious, such as making information objects computational entities and self-arranging, most others remain undiscovered.

2.4.3 Self-Sufficiency

For maximum autonomy, the default configuration for buckets is to contain all their code, data, user and password files, and everything else they need physically inside

the buckets. Optimizations exist in which code and password files can be “factored” out of the buckets, resulting in storage savings and easier management. However, these savings come at the cost of decreased autonomy and mobility (see below). Given proper tools, it is expected self-sufficient buckets can be easily managed and the increased storage overhead is negligible given that storage is “cheap”.

As for data, buckets can store data physically inside the bucket, or simply store “pointers” to data objects that exist outside the bucket. Internal data storage is preferred, however some data (e.g., database views) make sense to store as dynamic pointers. Although buckets can store either physical copies or pointers, buckets obviously can make no guarantees about the long-term survivability of items that lie outside the bucket. Buckets should provide the mechanisms to implement the internal vs. external data storage policies for specific applications.

2.4.4 Mobility

Related to self-sufficiency, buckets can be mobile. That is, they can physically move from place to place since they contain all the code and support files they need. Furthermore, placing a bucket on a host should require no modifications to the http server. For technical reports and re-prints, the need for mobility is not obvious, beyond the role that it plays in assisting information refreshing. However, an application proposed by the Air Force illustrates the power of mobility. In their plan, buckets are used to represent people and the buckets store supplemental human resources (HR) information (papers published, personnel reviews, CV materials, etc.) As people move between Air Force installations, their HR bucket moves with them, “plugging-into” the HR system at the host installation.

Mobility can be used in other situations where we wish to move the buckets in response to a particular workflow model. Rather than requiring the bucket to be anchored in a particular spot, it would be possible for a bucket to travel from place to place, and be local to the system that it is sampling data from. After collecting data at its various

stops, it could then be moved to a location where it is visible to a DL, and be indexed and found by users.

2.4.5 Heterogeneity

A significant requirement for buckets is that they all do not have to look or act the same. It is possible for different installations to locally modify the buckets created at that site to reflect their specific publishing policy or take advantage of known characteristics of the data they store. Similarly, it is possible for buckets to evolve differently over time, with new methods being added, deleted or overridden as appropriate. Furthermore, it should be possible to publish buckets with entirely different structure and functionality, based on what discipline the buckets support. Intuitively, an earth science bucket and a biomedical engineering bucket should at least have the option of looking and acting differently. However, buckets should retain enough basic methods so their version and features can be dynamically discovered.

2.4.6 Archive Independence

To the extent reasonable, buckets should work with any type of archive and/or DL. Similarly, they should not break any archive or DL. In fact, archives and DLs are not required for bucket operation. It should be possible for buckets to be indexed in any number of DLs. Archives, DLs, search engines, etc. are not intrinsic to the operation of buckets – they are add on services that can be used in management and resource discovery and should be completely decoupled from the buckets themselves.

CHAPTER THREE

BUCKET ARCHITECTURE

3.1 Overview

A bucket is a storage unit that contains data and metadata, as well as the methods for accessing both. It is difficult to overstress the importance of the aggregation design goal. In our experience with other NASA DLs, data was often partitioned by its semantic or syntactic type: metadata in one location, PostScript files in another location, PDF files in still another location, etc. Over time, different forms of metadata were introduced for different purposes, the number of available file formats increased, the services defined on the data increased, new information types (software, multimedia) were introduced, the logging of actions performed on the objects became more difficult. The result of a report being “in the DL” eventually represented so much DL jetsam - bits and pieces physically and logically strewn across the system. We responded to this situation with extreme aggregation.

The first focus of the aggregation was for the various data types. Based on experience gained while designing, implementing and maintaining LTRS and NTRS, we initially decided on a two-level structure within buckets:

- buckets contain 0 or more *packages*
- packages contain 0 or more *elements*

Actual data objects are stored as elements, and elements are grouped together in packages within a bucket. In LTRS and NTRS, a two-level architecture was sufficient for most applications, so this two-level architecture was retained as a simplifying assumption during bucket implementation. Future work will implement the semantics for describing arbitrarily complex, multi-level data objects.

An element can be a “pointer” to another object: another bucket, or any other arbitrary network object. By having an element “point” to other buckets, buckets can logically contain other buckets. Although buckets provide the mechanism for both internal and external storage, buckets have less control over elements that lie physically outside the bucket. However, it is left as a policy decision to the user as to the appropriateness of including pointers in an archival unit such as a bucket. Buckets have no predefined size limitation, either in terms of storage capacity, or in terms of number of packages or elements. Buckets can use a CNRI handle, a URN implementation, for a globally unique id. Buckets are accessed through 1 or more URLs. For an example of how a single bucket can be accessed through multiple URLs, consider two hosts that share a file system:

```
http://host1.foo.edu/bar/bucket-27/
http://host2.foo.edu/bar/bucket-27/
```

Both of these URLs point to the same bucket, even though they are accessed through different hosts. Also, consider a host that runs multiple http servers:

```
http://host1.foo.edu/bar/bucket-27/
http://host1.foo.edu:8080/bucket-27/
```

If the http server running on port 8080 defines its document root to be the directory “bar”, then the two URLs point to the same bucket.

Elements and packages have no predefined semantics associated with them. Authors can model whatever application domain they desire using the basic structures of packages and elements. One possible model for bucket, package, and element definition is based on NASA DL experiences. In Figure 4, packages represent semantic types (manuscript, software, test data, etc.) and elements represent syntactic representations of

the packages (a `.ps` version, `.pdf` version, `.dvi` version, etc.). Other bucket models using elements and packages are possible. For example, we have used buckets for entire research projects (Fig. 6) and university classes (Fig. 7) as well as for STI publications. Though the display of the two buckets is different, the two-level architecture of packages and elements is evident.

Buckets have the capability of implementing different policies as well: one site might allow authors to modify the buckets after publishing, and another site might have buckets be “frozen” upon publication. Still another site might define a portion of the bucket to receive annotations, review, or contributions from the users, while keeping another portion of the bucket frozen, or only changeable by authors or administrators. Buckets provide mechanism, not policy.

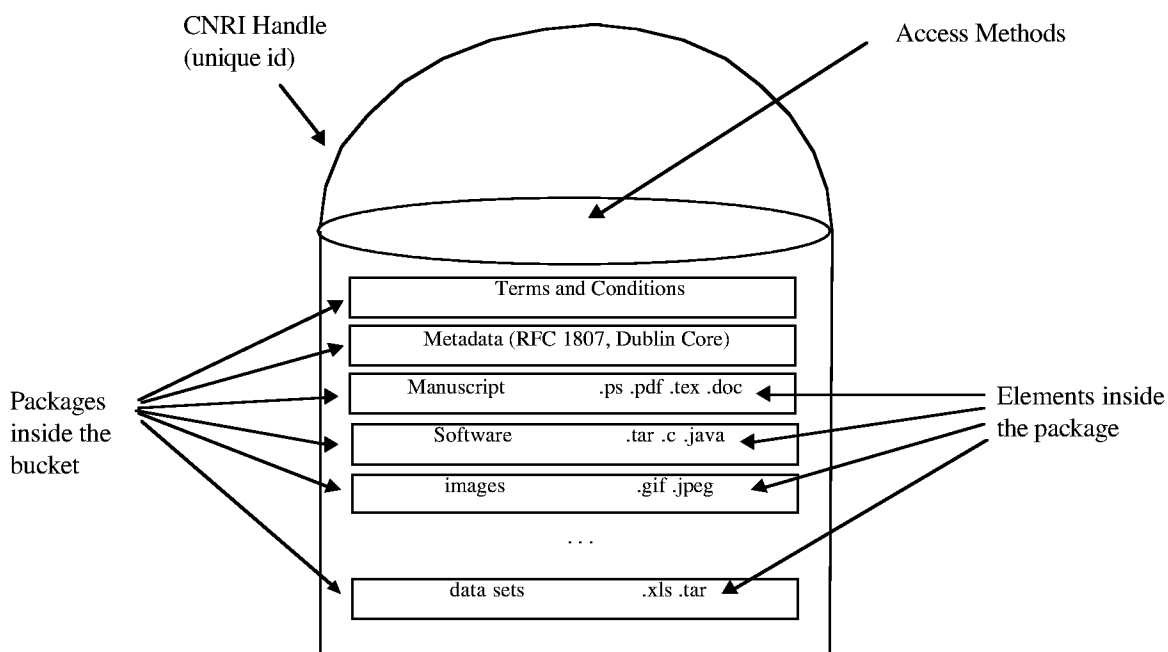


FIG. 5. Model of a typical NASA STI bucket.

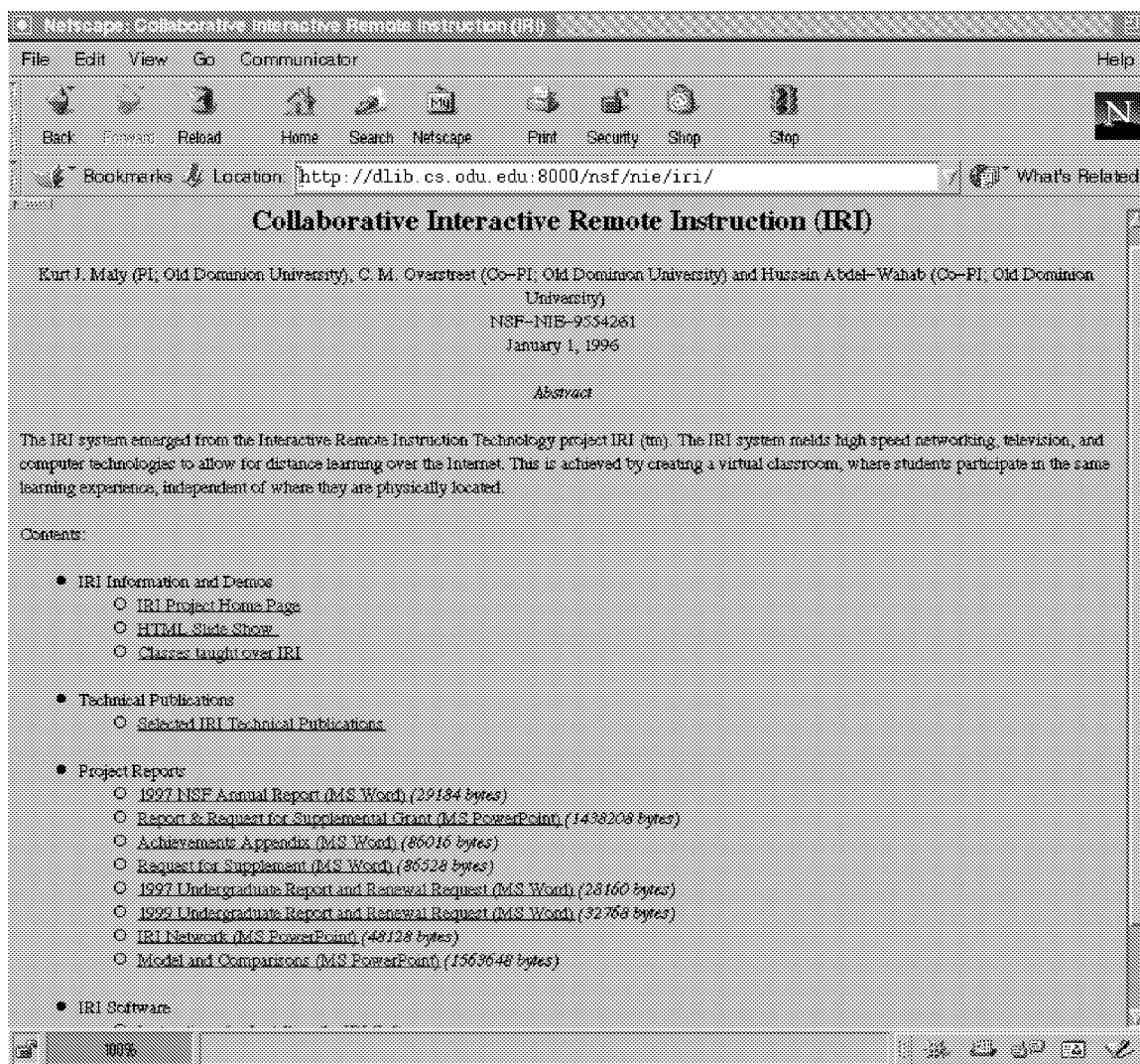


FIG. 6. Sample project bucket.

Another focus of aggregation was including the metadata with data. In previous experiences, we found that metadata tended to “drift” over time, becoming decoupled from the data it described or “locked” in specific DL systems and hard to extract or share with other systems. For some information types such as reports, regenerating lost metadata is possible either automatically or by inspection. For other information types such as experimental data, the metadata cannot be recovered from the data. Once the metadata is lost, the data itself becomes useless. Also, we did not want to take a proscriptive stance on metadata. Although the bucket itself has to ultimately chose one

metadata format as canonical for storing and modifying its internal structure information, buckets needed to be able to accommodate multiple metadata formats. Buckets do this by storing metadata in a reserved package and using methods for reading and uploading new metadata formats as elements in the metadata package. As a result, buckets can accommodate any number of past, present or future metadata formats.

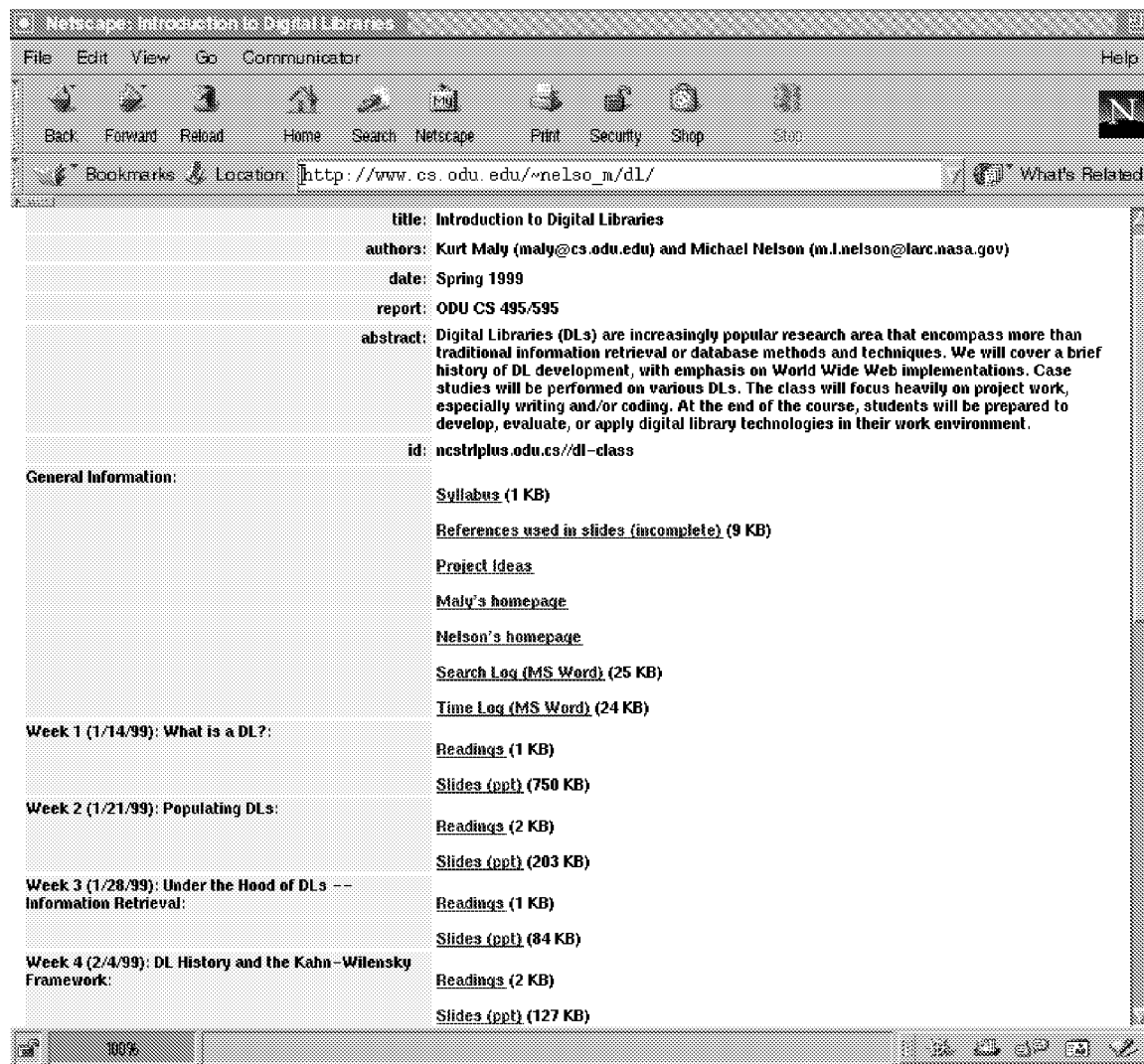


FIG. 7. Sample course bucket.

The final aggregation focus was on the services defined on buckets and the results of those services. The default state is for everything the bucket needs to display, disseminate, and manage its contents is contained within the buckets. This includes the

source code for all the methods defined on the bucket, the user ids and passwords, the access control lists, the logs of actions taken on the bucket, Multipurpose Internet Mail Extensions (MIME) (Borenstein & Freed, 1993) definitions and all other supporting technologies necessary for the bucket to function. The self-sufficiency and mobility design goals dictate that a bucket cannot make many assumptions about the environment that it will reside in and should require no server modifications to function.

3.2 Implementation

The buckets described in this chapter are version 1.6. Appendix A lists the full bucket history. Buckets are currently written in Perl 5 and use http as the transport protocol. However, buckets can be written in any language as long as the bucket API is preserved. Buckets were originally deployed in the NCSTRL+ project (Nelson, Maly, Shen, & Zubair, 1998), which demonstrated a modified version of the Dienst protocol (Lagoze, Shaw, Davis, & Krafft, 1995). Owing to their Dienst-related heritage, bucket metadata is stored in RFC-1807 format (Lasher & Cohen, 1995), with package and element information stored in NCSTRL+ defined optional and repeatable fields. Although buckets use RFC-1807 as their native format, they can contain and serve any metadata type. Dienst has all of a document's files gathered into a single Unix directory. A bucket follows the same model and has all relevant files collected together using directories from file system semantics. The bucket is accessible through a CGI script that enforces terms and conditions, and negotiates presentation to the WWW client.

Aside from Perl 5, http, and CGI, buckets make no assumptions about the environment in which they will run. Mobility is one of the design goals of buckets, and a corollary of that is that buckets should not require changes in a “reasonable” http server setup; where “reasonable” is defined to be allowance of the `index.cgi` convention. Once these assumptions have been met, buckets by default take care of everything themselves with no server intervention, including MIME typing, terms and conditions, and support libraries. Although bucket development was conducted under Solaris (Unix), buckets have been tested on a variety of system configurations (Table 1).

TABLE 1. System configurations used for bucket testing.

Architecture	Operating System	Perl	http server
Sparc	Solaris 2.7	5.005_03	Apache 1.3.9
Sparc	Solaris 2.7	5.005_03	NCSA httpd 1.5.2
Sparc	Red Hat 6.0 (Linux 2.2.5-15)	5.005_03	Apache 1.3.6
Intel x86	Windows NT 4.0 (1381 / SP 5)	Active Perl 5.005_03	Apache 1.3.12
Intel x86	Mandrake Linux 6.2	5.005_03	Apache 1.3.6
MIPS R10000	IRIX 6.5	5.004_04	Apache 1.3.4
RS/6000	AIX 4.2	5.002	Apache 1.3.12
PowerPC 604	Linux 2.0.33 (MkLinux)	5.004_01	Apache 1.2.6

The biggest difficulty in mobility across multiple platforms is locating the Perl interpreter. Buckets use the Unix-style “#!” construct to specify which interpreter should be used to process the script. For example, the first line in `index.cgi` script in bucket version 1.6 is:

```
#!/usr/local/bin/perl
```

Which explicitly specifies where Perl is expected to be found. On Unix systems, this is generally not a problem, since any of the following values are generally at least symbolic links to the canonical location of the Perl interpreter:

- `/usr/local/bin/perl`
- `/usr/bin/perl`
- `/bin/perl`

However, for Windows NT systems, Perl generally exists in a different location altogether, and the default Unix values are less likely to work. It is possible on the Windows NT version of Apache to bind the Perl interpreter to all scripts ending in `.cgi`, but for testing on our Windows NT system, the first line of the `index.cgi` script was changed to be:

```
#!/Perl\bin\perl.exe
```

For greater Unix portability there is a standard trick to gain slightly more portability. It is possible to replace the first line of the `index.cgi` script to contain:

```
#!/bin/sh -- # *- perl *-  
eval 'exec perl -S $0'  
if 0;
```

This invokes the Bourne shell and determines where Perl exists on the host by using the first value it finds in the `$PATH` environment variable. However, since this depends on the Bourne shell, it is even less likely to work on Windows NT systems than the current `#!` value. A general purpose bootstrapping procedure to specify the Perl interpreter has not been found.

3.2.1 Bucket Methods

Communication with buckets occurs through a series of bucket messages defined by the bucket API. The list of defined bucket methods is given in Table 2, and the bucket's detailed API is in Appendix B. Note that these are methods defined for our generic, all-purpose buckets. It is expected that local sites will add, delete and override methods to customize bucket structure details to their own requirements. It is important to note that regular users are not expected to directly invoke methods – the users require no special knowledge of buckets. All the user needs is the initial URL pointing to the bucket, and then the applicable methods for accessing its contents are automatically built into the bucket's HTML output. The other creation and management-oriented methods are expected to be accessed by a variety of bucket tools.

TABLE 2. Bucket API.

Method	Description
add_element	Adds an element to a package
add_method	Adds a method to the bucket
add_package	Adds a package to the bucket
add_principal	Adds a user id to the bucket
add_tc	Adds a T&C file to the bucket
delete_bucket	Deletes the entire bucket
delete_element	Deletes an element from a package
delete_log	Deletes a log file from the bucket
delete_method	Deletes a method from the bucket
delete_package	Deletes a package from the bucket
delete_principal	Deletes a user id from the bucket
delete_tc	Deletes a T&C file from the bucket
display	Displays and disseminates bucket contents
get_log	Retrieves a log file from the bucket
get_preference	Retrieves a preference(s) from the bucket
get_state	Retrieves a state(s) from the bucket
id	Displays the bucket's unique id
lint	Checks the buckets internal consistency
list_logs	Lists all the log files in the bucket
list_methods	Lists all the methods in the bucket
list_principals	Lists all the user ids in the bucket
list_source	List the method source
list_tc	Lists all the T&C files in the bucket
metadata	Displays the metadata for the bucket
pack	Returns a "bucket-stream"

set_metadata	Uploads a metadata file to the bucket
set_preference	Changes a bucket preference
set_state	Changes a bucket state variable
set_version	Changes the version of the bucket
unpack	Overlays a “bucket-stream” into the bucket
version	Displays the version of the bucket

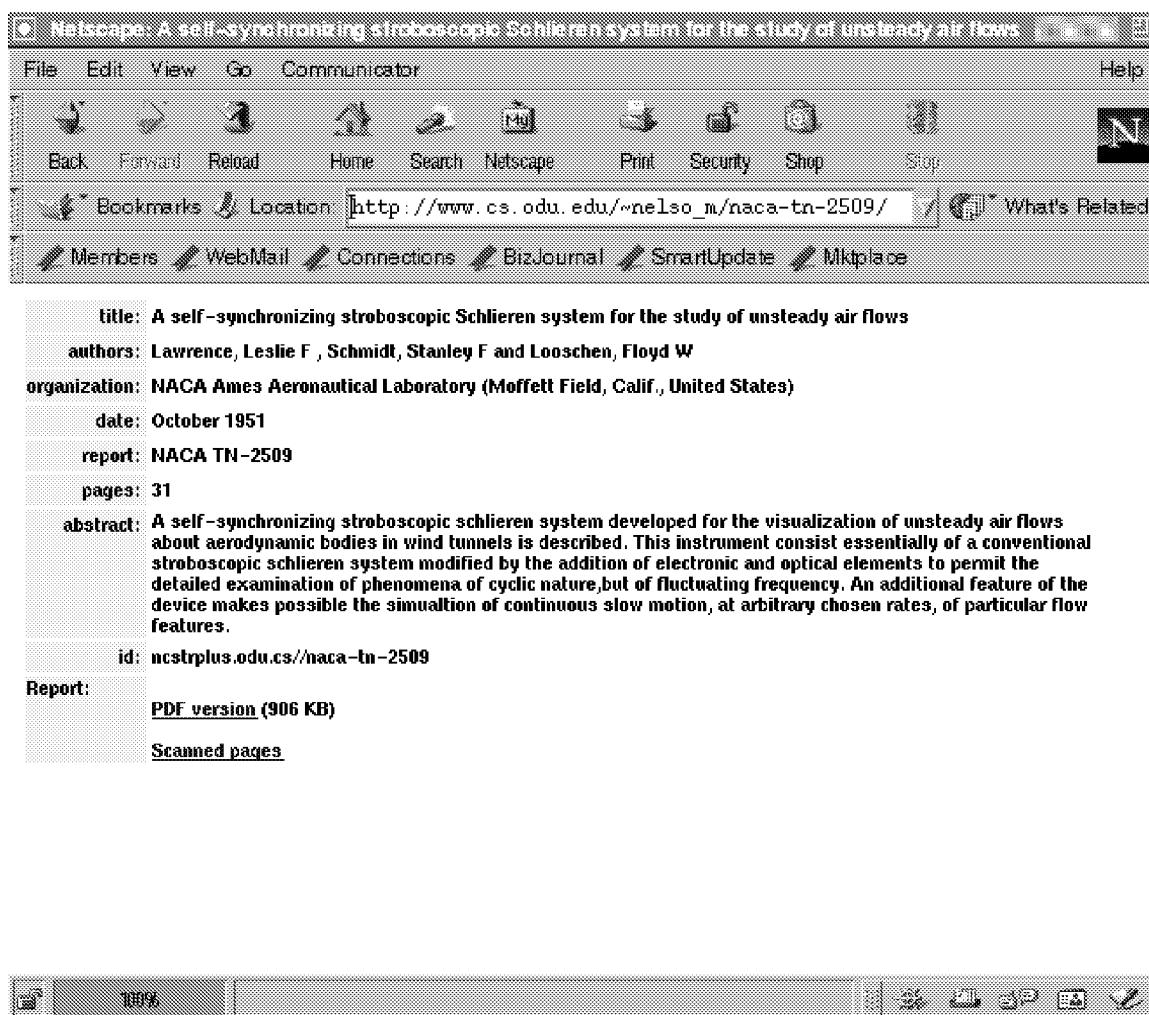


FIG. 8. Output of the “display” method.

Our reference implementation of buckets implements the bucket API using http encoding of messages. Buckets appear as ordinary URLs and casual users should not realize that they are not interacting with a typical web site. If no method is invoked via URL arguments, the “display” method is assumed by default. This generates a human-readable display of the bucket’s contents. For example, a bucket version of a NACA Technical Note can be reached at:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/
```

which is the same as:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/  
?method=display
```

Both of which will produce the output in figure 8. These URLs could be reached through either a searching or browsing function within a DL, or they could be typed in directly from above – buckets make no assumptions on how they were discovered. From the human readable interface the “display” method generates, if users wish to retrieve the PDF file, they click on the PDF link that was automatically generated:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/  
?method=display&pkg_name=report.pkg  
&element_name=naca-tn-2509.pdf
```

which would cause the WWW browser to launch the PDF reader application or plug-in. Similarly, if the users wished to display the scanned pages, selecting the automatically created link would send the following arguments to the “display” method:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/  
?method=display&pkg_name=report.pkg  
&element_name=report.scan
```

which would produce the output seen in figure 9. To the casual observer, the bucket API is transparent. However, if individual users or harvesting robots know a particular URL is actually a bucket, they can exploit this knowledge. For example, to extract the metadata in default (RFC-1807) format, the URL would be:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/
?method=metadata
```

which would return the metadata in a structured format, suitable for inclusion in an index being automatically built by a DL. If a user or agent wishes to determine that nature of a bucket, a number of methods are available. For example, to determine the bucket's version, the message is:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/
?method=version
```

And to see what methods are defined on a bucket:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/
?method=list_methods
```

However, if a harvester is not bucket-aware, it can still “crawl” or “spider” the bucket URLs as normal URLs, extracting information from the HTML human-readable interface generated by the “display” method (assuming the “display” method is not restricted by T&C). Buckets offer many expressive options to the users or services that are bucket-aware, but are transparent to those who are not bucket-aware.

3.2.2 File Structure

Buckets take advantage of the package/element construct for their internal configuration. In addition to the user data entered as packages and elements, the bucket keeps its own files as elements in certain reserved packages. Thus, methods such as “add_element”, “delete_element” and so forth can be used to update the source code for the bucket, update the password files, etc. Table 3 lists the predefined packages and some of the

elements they contain. By convention, these packages begin with an underscore (“_”) character. Figure 10 provides a model representation of the structure of a typical bucket, with internal packages and elements on the left and user-supplied data packages on the right.

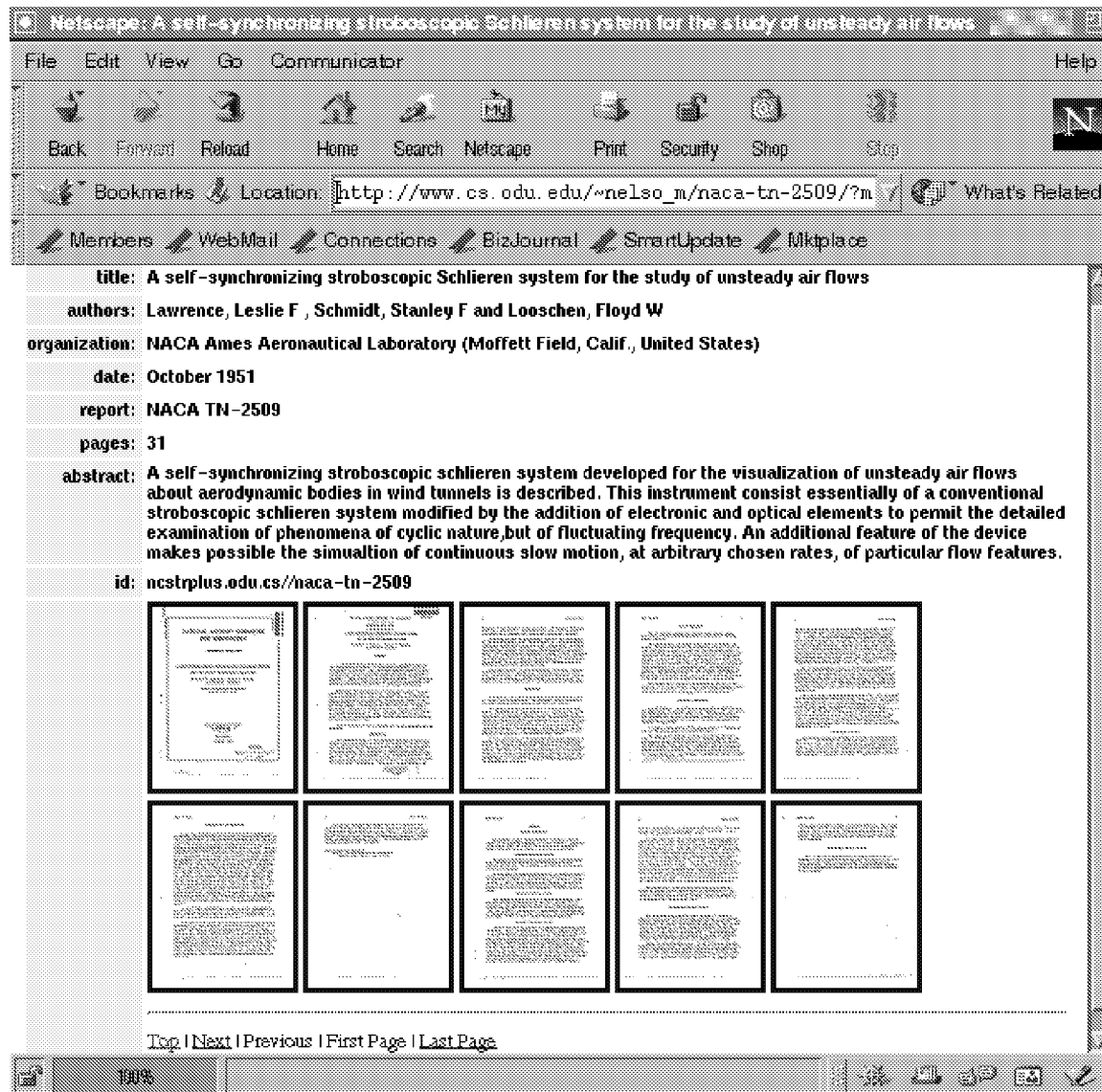


FIG. 9. Thumbnails in the “display” method.

TABLE 3. Reserved packages.

Package	Elements Within the Package
_http.pkg	cgi-lib.pl – Steven Brenner’s CGI library encoding.e – a list of MIME encoding types mime.e – a list of MIME types
_log.pkg	access.log – messages received by the bucket
_md.pkg	[handle].bib – a RFC-1807 bibliographic file other metadata formats can be stored here, but the .bib file is canonical
_methods.pkg	1 file per public method
_state.pkg	1 file per stored state variable
_tc.pkg	1 file per .tc (terms and condition) file password file .htaccess file

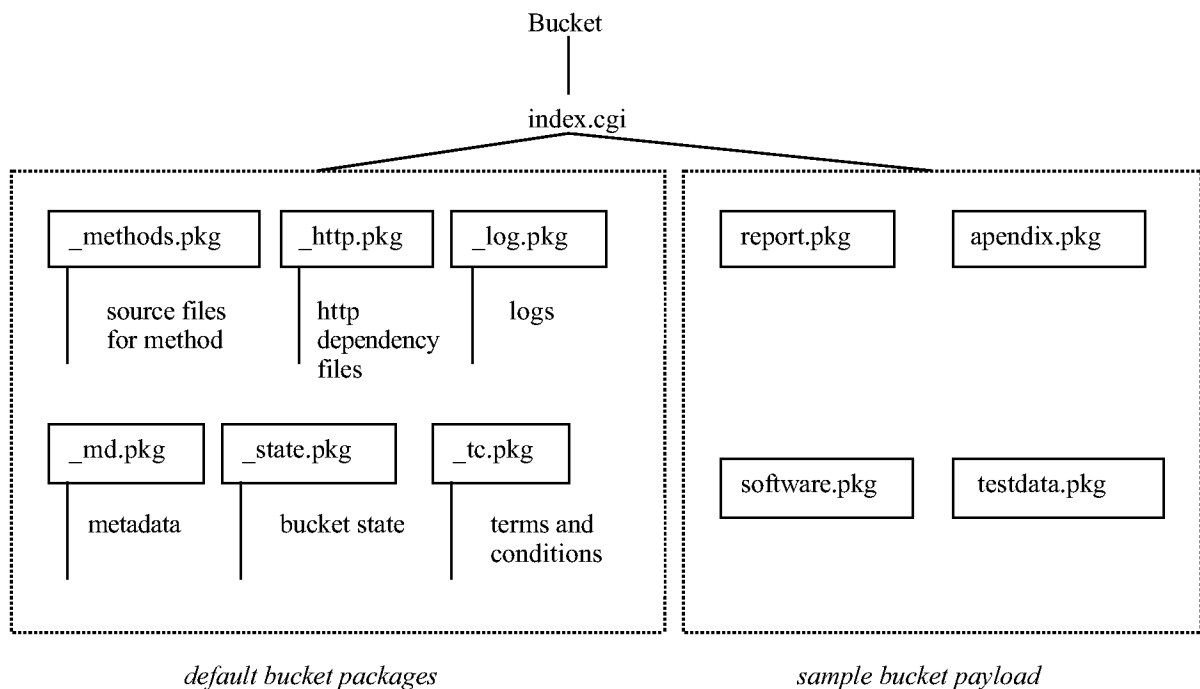


FIG. 10. Bucket structure.

3.2.3 Terms and Conditions

Bucket terms and conditions are currently implemented using http and CGI facilities. Although it is possible to access a bucket from the command line, this would effectively bypass any security measures implemented by the bucket. Buckets hosted on shared machines must use file system protections to prevent users from bypassing the bucket API. Building from the CGI facilities, buckets implement simple access control lists (ACLs) that restrict access based on username/password pairs, Internet Protocol (IP) hostnames, and IP addresses. It is also possible to apply these restrictions to entire methods, entire packages, or package/element pairs. The first example given below assumes the buckets are entirely self-contained and nothing has been factored out. For collection-wide T&C, factoring out the often repetitive T&C files as well as the user ids and passwords allows for easier maintenance. However, the bucket operation remains the same, only the location of where the bucket looks (internally vs. a shared location) is changed. Creation of user, hostname or address groups is supported as well. However, the bucket does not directly process these groups, it first flattens the groups to a list of values, and then they are processed normally. Factoring and group support are orthogonal and can be combined.

The ACLs are stored as elements (with a `.tc` extender) in the `_tc.pkg` package. The filename for the `.tc` file specifies which method it protects. For example, to protect the entire method “list_tc”, the file `_tc.pkg/list_tc.tc` could contain the following entries, or “directives”:

```
user:nelson
user:maly
host:*.larc.nasa.gov
host:*.cs.odu.edu
```

Which would require the http-based user authentication for either “nelson” or “maly” to be given, and for the originating computer to have either a `.larc.nasa.gov`

or a `.cs.odu.edu` IP address. To protect all elements of a package from being displayed, a `_tc.pkg/display.tc` file could contain:

```
addr: 128.155.*
package: data.pkg
```

This would require access to any of the elements within the package “data.pkg” to originate from a machine on the 128.155 subnet. Other packages and elements can be retrieved from any machine by any user. To restrict who can delete a specific element from, the file `_tc.pkg/delete_element.tc` would contain:

```
user:nelson
package:report.pkg
element:report.pdf
element:report.ps
```

This would prevent the two specified elements of the `report.pkg` package from being deleted by anyone other than the user “nelson”. Note that if in the above example, the `user:` line was deleted, and no other `user:`, `host:` or `addr:` line was present, this would have the same effect of preventing anyone from deleting the above elements. The `delete_element.tc` file would have to be changed before those elements could be deleted.

To specify T&C for the entire bucket, an `index.tc` file is used. An `index.tc` file uses the same syntax and can be used in conjunction with method T&C files. If `index.tc` and `add_package.tc` both exist, then calls to “add_package” would have to satisfy both T&C files, with `index.tc` being enforced first.

For ease of management, it is possible to define groups of users, hostnames and address. These bucket groups are not implemented with http groups, but rather as text files with the following syntax:

```
group1: user1 user2 user3
group2: user2 user4
```

Terms and conditions are enforced using CGI mechanisms. This includes checking the environment variables `REMOTE_HOST`, `REMOTE_ADDR`, and `REMOTE_USER`. The CGI environment on the http server automatically sets `REMOTE_HOST` and `REMOTE_ADDR`. `REMOTE_USER` is set when CGI user authentication is done. When a bucket method is invoked, before the bucket loads the appropriate code to execute that method, the bucket checks in the `_tc.pkg` package to see if there is a `.tc` file for the called method. If the `.tc` file exists, its contents are read. If `host:` or `addr:` lines are present, and if the package and element arguments passed in are also listed in the `.tc` file, then `REMOTE_HOST` and `REMOTE_ADDR` are compared with the `host:` and `addr:` lines for a match. If a match is not made, an error message is returned and execution halts. If a match is made, execution continues with no interruption.

Utilizing the username/password function (i.e., `REMOTE_USER`) is slightly more complicated. If the `.tc` files have a `user:` line, then current bucket execution is redirected. That is, if the display method requires `REMOTE_USER` to be set, then:

```
http://dlib.cs.odu.edu/bucket/?method=display
```

is redirected to:

```
http://dlib.cs.odu.edu/bucket/restricted/?method=display
```

Every bucket has a `/restricted/` redirect capability that invokes the CGI username / password authentication by means of a `.htaccess` file in the `restricted` directory. The `.htaccess` file lists the usernames that it will accept. The passwords are stored encrypted in a separate, Unix-style password file. All of this is kept in the `_tc.pkg` package. If the user authenticates to a recognized username, and that username is one that is required in the `.tc` file, then execution continues. If not, an error message is generated and execution halts.

To further illustrate the operation of bucket T&C, consider an example where we wish to restrict all access to a large number of buckets so that only `*.cs.odu.edu` and `*.larc.nasa.gov` machines can access them. Since all the buckets will have the same T&C profile, we will use both factoring and groups for easier management of the T&C files in the buckets and the specification. The buckets' preferences (discussed in the next section) will be changed so they look for their `.tc` files outside the bucket. The `tc_server` preference would be changed from `internal` to a file system location such as:

```
/usr/local/buckets/tc/group-10/
```

The bucket would look inside this directory, and see the presence of an `index.tc` file that would specify T&C for the entire bucket – all methods. The `index.tc` file would have contents similar to:

```
host_group: hgroup-10
```

If the buckets' `host_group` preference has been changed from `internal` to:

```
/usr/local/buckets/tc/
```

Then the file `/usr/local/buckets/tc/host_group` would have entries similar to:

```
hgroup-1: blearg.larc.nasa.gov lion.cs.odu.edu
hgroup-2: jaguar.cs.odu.edu *.nasa.gov
...
hgroup-10: *.larc.nasa.gov *.cs.odu.edu
...
```

The bucket preferences and the values in the various `.tc` files and group files are all manageable from the Administration Tool (chapter seven). While the T&C mechanism described above is sufficient for a large number of applications, they are not rich enough

for a full array of needs anticipated for NASA DLs. In addition, the terms and conditions can be defeated if the values of `REMOTE_HOST`, `REMOTE_ADDR`, or `REMOTE_USER` are forged.

Table 4 lists all the directives that T&C files recognize. The `inform:` directive, although present in the T&C file, does not specify access control. Instead, it specifies who should be informed when an action takes place. If an action is successful, the bucket generates an email message informing the recipient that the action was successfully completed. If the action was not successful, the bucket generates an email message indicating the failure of the action along with an HTML form that will allow the recipient to reattempt the action. The `inform:` directive can be used in conjunction with other directives.

TABLE 4. Directives available in T&C files.

Directive	Arguments
<code>user:</code>	principal names
<code>group:</code>	principal groups
<code>addr:</code>	IP address (can be a Perl regular expression)
<code>addr_group:</code>	IP address groups
<code>host:</code>	IP hostname (can be a Perl regular expression)
<code>host_group:</code>	IP hostname groups
<code>package:</code>	name of a package in this bucket
<code>element:</code>	name of an element in this package (must be used with a <code>package:</code> directive)
<code>inform:</code>	email addresses

3.2.4 Internal Bucket Operation

In this section, we examine what happens internally when a CGI-based bucket receives a message. The details are specific to the current implementation of Perl-based buckets; other implementations are free to implement the bucket API differently.

A user contacts a http server and specifies the bucket they wish to communicate with. This can be done by either explicitly naming the `index.cgi`:

```
http://foo/bar/bucket/index.cgi
```

or implicitly:

```
http://foo/bar/bucket/
```

The CGI parsing is done with `cgi-lib.pl` (Brenner, 2000) that is stored inside the bucket in the `_http.pkg` package. `CGI.pm` (Stein, 1998) is not used for two reasons: 1) it is not part of the standard Perl library, so it would have to be carried in the bucket as well; 2) it did not perform well as well as `cgi-lib.pl` in command line operation (used frequently in testing and debugging). The `index.cgi` script parses the input string to determine which method the user is requesting. If no method is specified, the display method is assumed. The `index.cgi` script then looks in the `_tc.pkg` package to see if an `index.tc` or `method.tc` file exists (where `method` is the name of the method that is being invoked). If either (or both) file(s) exists, the T&C are enforced as described in the previous section. If the T&C are satisfied, the `index.cgi` script then performs a run-time include of the source code of the method. The `index.cgi` script then calls the function with the same name as the method invoked, which is assumed to be defined in the source code included at run-time. This way only the code for the invoked method is accessed. This procedure also allows for the `index.cgi` script to be written so that it makes no assumptions about the methods that are available, allowing methods to be added and deleted for specific bucket instantiations. This entire process can be encapsulated in

this Perl code snippet, where `$method` is a variable containing the name of the requested method:

```
$method_file = "$method_dir/$method.pl";
if (-f $method_file) {
    &tc($method);          # check tc
    # if we made it out of &tc, we must be ok...
    require "$method_file"; # run-time include
    &$method;              # calls the method
}else {
    # method not found in bucket
    &unsupported($method);
}
```

If the “display” method is called with specific package and element arguments, then the named file is returned. However, this is not done through “normal” http operations – to enforce data hiding, packages have `.htaccess` files that prevent any direct access of their elements. The `index.cgi` script opens the file for reading, sets the correct MIME type by checking the element `_http.pkg/mime.e`, and then writes the file to STDOUT. If the file being returned to the user is an HTML file, the relative URLs are re-written to access elements within the bucket. This is necessary because of the inherent conflict between URLs, which are tightly tied with file location, and the bucket’s data hiding, which prevents access of specific file locations. If the element being requested by the “display” method call is a URL to a location outside of the bucket, the bucket will log that a “display” call was made, where the intended location is, and then issue an http status code 302 (redirect) to the client.

3.2.5 Metadata Extensions

The metadata file in a bucket plays an extremely important role. Not only does it hold the traditional bibliographic citation material, it also encodes the structure of the bucket’s contents. This structure is read and processed when the bucket’s “display” method is called and the bucket reveals its structure in a human readable, HTML format.

RFC-1807 is an extensible format. To describe the two-level bucket structure, two tags have been defined: “PACKAGE::” and “ELEMENT::”. All previously defined RFC-1807 tags are also available with the “PACKAGE” and “ELEMENT” prefix: “PACKAGE-TITLE::”, “ELEMENT-END::”, etc. Currently only the values for the “PACKAGE-TITLE::” and “ELEMENT-TITLE::” tags are revealed during a “display” method call, however this is likely to change in the future. Figure 11 shows the RFC-1807 metadata for a bucket:

```

BIB-VERSION:: X-NCSTRL+1.0
ID:: ncstrplus.odu.cs//naca-tn-2509
TITLE:: A self-synchronizing stroboscopic Schlieren system for the study of
unsteady air flows
REPORT:: NACA TN-2509
AUTHOR:: Lawrence, Leslie F
AUTHOR:: Schmidt, Stanley F
AUTHOR:: Looschen, Floyd W
ORGANIZATION:: NACA Ames Aeronautical Laboratory (Moffett Field, Calif.,
United States)
DATE:: October 1951
PAGES:: 31
ABSTRACT:: A self-synchronizing stroboscopic schlieren system developed
for the visualization of unsteady air flows about aerodynamic bodies in
wind tunnels is described. This instrument consist essentially of a
conventional stroboscopic schlieren system modified by the addition of
electronic and optical elements to permit the detailed examination of
phenomena of cyclic nature, but of fluctuating frequency. An additional
feature of the device makes possible the simulation of continuous slow
motion, at arbitrary chosen rates, of particular flow features.

PACKAGE:: report.pkg
PACKAGE-TITLE:: Report
ELEMENT:: naca-tn-2509.pdf
ELEMENT-TITLE:: PDF version
ELEMENT-END:: naca-tn-2509.pdf

ELEMENT:: report.scan
ELEMENT-TITLE:: Scanned pages
ELEMENT-END:: report.scan
PACKAGE-END:: report.pkg

PACKAGE:: staff.pkg
PACKAGE-TITLE:: For LaRC Staff
ELEMENT:: report.tiffs
ELEMENT-TITLE:: TIFFs
ELEMENT-END:: report.tiffs

ELEMENT:: maintenance.html
ELEMENT-TITLE:: Maintenance page
ELEMENT-END:: maintenance.html
PACKAGE-END:: staff.pkg

END:: ncstrplus.odu.cs//naca-tn-2509

```

FIG. 11. RFC-1807 metadata.

The values for “PACKAGE::”, “PACKAGE-END::”, “ELEMENT::” and “ELEMENT-END::” correspond to the actual filesystem names inside the bucket. Just

as elements can only exist within packages, “ELEMENT” tags and prefixed tags must be contained within their respective “PACKAGE::” / “PACKAGE-END::” tag pairs.

3.3 Discussion

The previous sections describe the “normal” operations of a bucket. However, a bucket’s operation can be transformed by the setting of bucket preferences. In this section, we list what is possible through bucket preferences, as well as examining a number of systems issues with the current bucket implementation.

3.3.1 Bucket Preferences

Bucket preferences can be checked and set through the “get_preference” and “set_preference” methods, respectively. Preferences allow individual buckets to tailor their operation to reflect their unique requirements, but yet retain a standard, public way of being changed in the future. Table 5 lists the currently defined preferences and gives a short explanation of their function.

Inspection of Table 5 will reveal that many preferences exist so the method source code, user names and passwords, and T&C files can all be “factored out” of the bucket. The default model is the bucket carries all of this internally, thus allowing for greater mobility and independence. However, this level of freedom comes at the cost of increased storage and complexity in managing multiple copies of source code, passwords, etc. So we provide the mechanism to factor out all the pieces that do not need to be internally stored in the bucket. However, these need not be permanent decisions – a mostly homogenous collection of buckets can all share from a central store their source code and other items that have been removed. However, specific buckets that require different functionality or a higher level of independence can have their preferences changed so they return to the default model of internal storage for some or all things.

TABLE 5. Bucket preferences.

Preference	Default Value	Description
access.log	on	This is the name of the single default log, and by default logging is set to “on”. Logging can be turned off by setting this value to “off”.
addr_group	internal	By default, the bucket expects to internally store the file that maps addr_group names to lists of IP addresses. Filesystem pathnames are other acceptable values.
bcs_server	<i>(none)</i>	A bucket can choose which Bucket Communication Space server it communicates with. The current default is just a sample value, and is likely to be site dependent. URLs are acceptable values here.
expanding	off	The bucket display by default lists all elements in all packages at once. By setting this preference to “on”, the elements will not be visible until the package name is “clicked”, revealing its contents.
framable	off	By default, the bucket “display” method includes JavaScript to keep the bucket from being “trapped” inside a frame. Setting this preference to “on” allows buckets to exist inside frames.
group	internal	By default, the bucket expects to internally store the file that maps group names to lists of user names. Filesystem pathnames are other acceptable values.
host_group	internal	By default, the bucket expects to internally store the file that maps host_group names to lists of IP hostnames. Filesystem pathnames are other acceptable values.

maxdata	5000000	This is the default value for maximum file size of an uploaded file. Any integer value greater than or equal to zero is acceptable.
method_server	internal	By default, the bucket expects to find the source code for methods inside the bucket. Filesystem pathnames are other acceptable values.
passwd	internal	By default, the bucket expects to find the password file (stored in Unix “/etc/passwd” format) inside the bucket. Filesystem pathnames are other acceptable values.
sfx_server	(none)	The location of a Special Effects (SFX) reference linking server. This value is just a placeholder; the nature of SFX insures that this needs to be set to a site-specific value. URLs are acceptable values.
tc_server	internal	By default, the bucket expects to internally store the T&C files for the bucket methods. Filesystem pathnames are other acceptable values.
thumbnail_increment	10	When displaying thumbnails of scanned pages, this preference determines how many thumbnails to show at a time. Any integer greater than 1 is an acceptable value.

3.3.2 Systems Issues

There are a number of systems-related issues concerning the current Perl implementation of buckets, which might not be present in alternate bucket implementations. These issues include: interaction between buckets and http caching; http server permissions and file permissions; and resource consumption by buckets.

Because they depend on CGI, client or server http caches should automatically not store responses to bucket messages. While this can result in lower performance for

the user for repeated access to DL objects, given the potentially dynamic nature of buckets, not caching responses is desirable.

Another common issue in bucket operation is that the owner of the files that make up the bucket and the user id of http server do not always match. Since it is possible to change the bucket through bucket methods, the http server needs to be able to add, delete and modify files that make up the bucket. There are four ways to accomplish this:

- The files that comprise the bucket can be world writable. While this allows the http server to write to the bucket, it also makes the bucket vulnerable to anyone with file system access to the bucket. This method would only be reasonable if the interactive logins on the machine hosting the buckets were limited to trusted parties.
- The `index.cgi` script can be `setuid`, so when it is invoked, it runs as the owner of the script, not the caller of the script (in this case, the http server). However, for security purposes on general-purpose machines many system administrators do not allow `setuid` programs outside of file partitions used by the operating system. Furthermore, many operating system kernels have a potential security flaw via a race condition in invoking `setuid` scripts, so the Perl interpreter will not run `setuid` scripts unless the kernel has been patched or the scripts have been wrapped with a C program (Wall, Christiansen, & Schwartz, 1996).
- The http server can be run as a user (or group member) that has write access to the files in the bucket. However, most http servers on the standard port (80) are run as “nobody” or some other account with minimal privileges and no interactive login. However, it is possible to run a http server on a non-standard port that is run as the owner of the files in the bucket. This will work, but it does leave open the possibility that if attackers were to compromise the http server, they could gain access to a privileged account and not a limited one such as “nobody”.

- Current versions of Apache, a popular free-source http server, have a “setuid module”. This allows the installer of apache to decide if all CGI programs should run not as the same owner of the http server, but as the owner of the CGI file. This is an elegant, general solution if a site is running an http server with this capability.

Note that buckets make no assumptions how the problem of http user id and bucket file user id is solved – only that it is solved. If an http server does not have write permission to a bucket’s files, attempts to update the bucket will fail. Assuming file system permissions permit, read attempts will continue to function normally.

The current implementation of buckets should still be considered research prototypes. As such, they consume storage resources more greedily than a stable, production version would. Bucket version 1.6 currently requires 68 inodes and 144 kilobytes storage for an “empty” bucket. Inodes are used by the Unix filesystem to store information on individual files and directories. Inodes are finite, but additional inodes can be allocated by a systems administrator. The inode and kilobyte requirements of the current implementation are a non-trivial overhead imposed by the buckets. However, there are other factors to consider:

- These are research prototypes, and as such are “wasteful” in the name of convenience. The source code has full documentation and other features not required for use in a non-development setting. Many of the inodes are consumed to store simple preferences (e.g., “on” or “internal”) where in a production system these could be compressed into a single file or data-structure. Although such optimization has not been vigorously pursued (see chapter six for optimization performed for the UPS project), it is anticipated that 50% of the inodes and 30% of the kilobytes required could be reduced.
- Furthermore, the storage requirement is small when compared with the large aggregations of data that they are designed to hold. For example, in the

NACATRS digital library, the average storage requirement per scanned page is approximately 80KB (Nelson, 1999). Thus, the KB required for a bucket is less than two scanned pages. 144KB should not be an issue when using buckets to store 100 scanned page reports, potentially with large supporting data sets or software.

- Storage is cheap and getting cheaper. Lesk (1997) reports that storage is about 4.5 MB / US \$1.0. A quick glance through a current *Computer Shopper* (a popular computer hardware mail order retailer) reveals an average of about 20-25 MB / US \$1.0, which fits the profile of storage capacity doubling roughly every 1.5 years. The exact numbers are not as important as the trend: with each refresh or migration, the bucket storage overhead problem will decrease relative to the amount of storage available at a fixed price.
- For DL applications where buckets are likely to be largely homogeneous, factoring of source code, T&C, and authentication information is available to reduce the inode and kilobyte requirements. For example, factoring out just the method source code of a version 1.6 bucket can save 31 inodes and 67 kilobytes.

So while it is true that buckets do impose additional storage requirements, it is felt that the small additional cost is more than offset by the additional capabilities that buckets provide.

CHAPTER FOUR

DUMB ARCHIVES

4.1 Overview

Buckets are the smart objects in the Smart Object, Dumb Archive DL model. To complement the buckets, dumb archives exist primarily to aid in the discovery and group management of buckets. It is possible to use buckets in other DL models, but SODA provides the most striking demonstration of the shift in responsibilities.

4.1.1 The SODA DL Model

We present a model that defines DLs as composed of three strata (Fig. 12):

- *digital library services* - the "user" functionality and interface: searching, browsing, usage analysis, citation analysis, selective dissemination of information (SDI), etc.
- *archive* - managed sets of digital objects. DLs can poll archives to learn of newly published digital objects, for example.
- *digital object* - the stored and trafficked digital content. These can be simple files (e.g., PDF or PS files), or more sophisticated objects such as buckets.

DLs are built by Digital Library Service Providers (DLSPs) that:

- identify a user group
- identify archives holding buckets of interest and individual bucket owners
- negotiate terms and conditions with publishing organizations (archive and individual bucket owners)
- create indices of appropriate subsets through extracting bucket metadata
- create DL services such as search, browse, and reference linking
- create user interaction services such as authentication and billing

In most DLs, the digital library services (DLS) and the archive functionality are tightly coupled. A digital object is placed in an archive, and this placement uniquely determines in which DL it appears. We believe that if there is not a 1-1 mapping between archives and DLs, but rather a N-M mapping, the capacity for interoperability is greatly advanced. A DL can draw from many archives, and likewise, an archive can contribute its contents to many DLs.

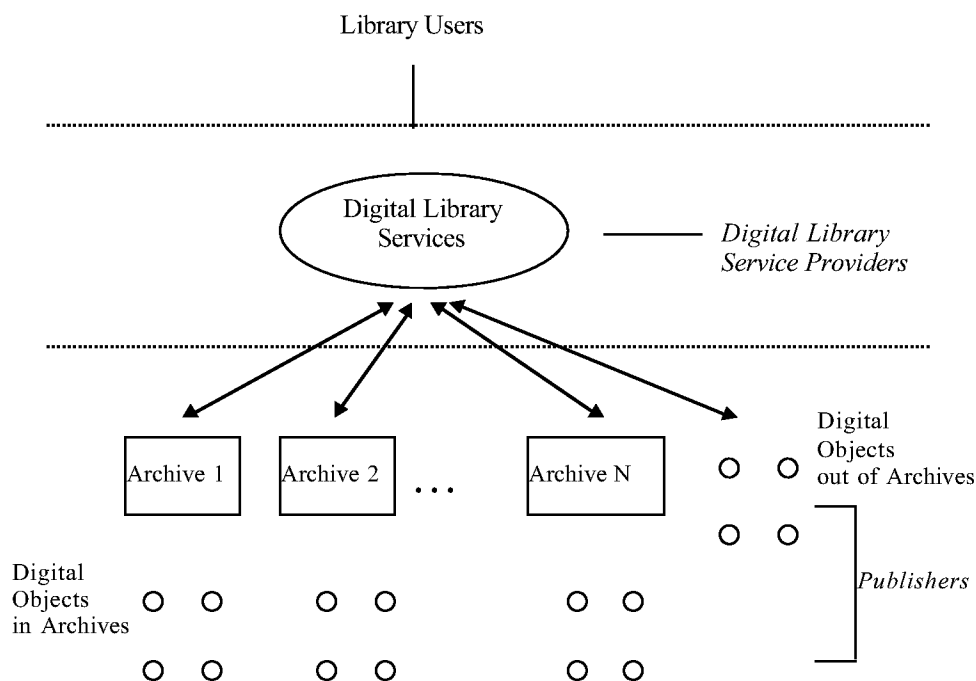


FIG. 12. The three strata of DLs

However, since we can no longer be sure which DL will be used for the discovery and presentation of an object, it is necessary to evolve the notion of the object and to imbue it with greater functionality and responsibility. DL objects should be self-sufficient, intelligent, and aggregative and capable of enforcing their own terms and conditions, negotiating access, and displaying their contents.

Much of the traditional functionality associated with archives (terms and conditions, content display, etc.) has been “pushed down” into the objects, making the objects “smarter” and the archives “dumber”. To demonstrate a SODA DL, a reference

implementation, NCSTRL+ (fully described in Chapter Six), has been constructed which implements each of the 3 strata listed above using the Dienst protocol and http services. The DLSs are provided by using the basic core of Dienst for searching, browsing and similar services. The archive functionality was originally implemented using a modified version of Dienst, because a bucket-based archive system was not originally available.

The observation that motivates the SODA model for DLs is that digital objects are more important than the archives that hold them. Many DL systems and protocols are reaching a point where DL interoperability and object mobility are hindered by the complexity of the archives that hold the objects. The goal of the current work is to increase the responsibilities of objects, and decrease the responsibilities of archives. If digital objects themselves handle presentation, terms and conditions and their own data management, it will be easier to achieve interoperability between heterogeneous DLs as well as increase object mobility and longevity. As a consequence, more DLSPs should be encouraged to build digital libraries for various user communities.

4.1.2 Archive Design Space

Archives exist primarily to assist DLs in locating objects -- they are generally not for direct user access. It appears that many digital libraries and their associated access protocols (e.g., Dienst and the Repository Access Protocol (RAP) (Lagoze & Ely, 1995)) have become unnecessarily complex. For example, the Dienst protocol contains a built-in document object model, and this limits its applicability in different domains and makes it more difficult to transition to evolving document object models. It is the archived objects, not archives, that should be responsible for the enforcement of terms and conditions, negotiation and presentation of content, etc. Although it is expected that some archive implementations will retain portions of the above functionality -- indeed, SOSA (Smart Objects, Smart Archives) may become the most desirable DL model -- a “dumb archive” model is used here to illustrate the full application of smart objects (buckets). When archives become “smart” again, it will with other functionalities, not duplication of bucket

functionality. Using this terminology, Table 6 illustrates how the archive design space partitions.

TABLE 6. The archive design space.

	Smart Archives	Dumb Archives
Smart Objects	SOSA: Smart Objects, Smart Archives DL Example: none known	SODA: Smart Objects, Dumb Archives DL Example: NCSTRL+
Dumb Objects	DOSA: Dumb Objects, Smart Archives DL Example: NCSTRL	DODA: Dumb Objects, Dumb Archives DL Example: any anonymous FTP server with .ps.Z files

4.1.3 Publishing in the SODA Model

Separating the functionality of the archive from that of the DLS allows for greater interoperability and federation of DLs. The archive's purpose is to provide DLs the location of buckets (the DLs can poll the buckets themselves for their metadata), and the DLs build their own indexes. And if a bucket does not “want” to share its metadata (or contents) with certain DLs or users, its terms and conditions will prevent this from occurring. For example, it is expected that the NASA digital publishing model will begin with technical publications, after passing through their respective internal approval processes, to be placed in a NASA archive. The NASA DL (which is the set of the NASA buckets, the NASA archive(s), the NASA DLS, and the user communities at each level) would poll this archive to learn the location of buckets published within the last week. The NASA DL could then contact those buckets, requesting their metadata. Other DLs could index NASA holdings in a similar way: polling the NASA archive and contacting the appropriate buckets. The buckets would still be stored at NASA, but they could be indexed by any number of DLs, each with the possibility for novel and unique

methods for searching or browsing. Or perhaps the DL collects all the metadata, then performs additional filtering to determine applicability for inclusion into their DL. In addition to an archive's holdings being represented in many DLs, a DL could contain the holdings of many archives. If all digitally available publications are viewed as a universal corpus, then this corpus could be represented in N archives and M DLs, with each DL customized in function and holdings to the needs of its user base. Figure 13 illustrates the SODA publishing model.

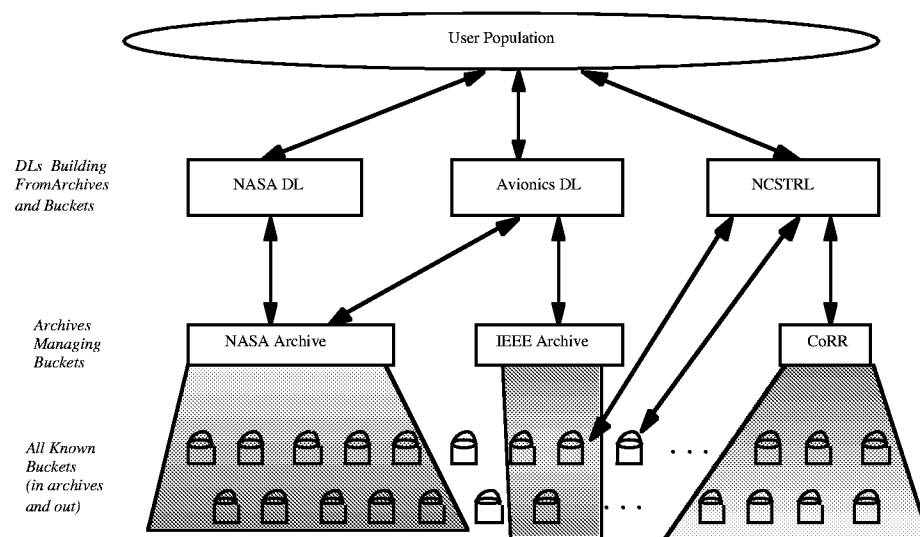


FIG. 13. The SODA publishing model.

4.2 Implementation

A few simple prototypes of a DA were built as standalone services, but eventually the decision was made to extend an existing bucket with new methods so it could function as the DA. Not only did this allow for rapid development of the DA, but it also showcases the flexibility in modifying buckets for different purposes. It should also be noted that although DA was created to keep track of buckets, there is nothing in its implementation that requires the objects it tracks to be buckets. For example, it would be possible to use DA for an archive of PDF files.

4.2.1 Implemented Methods

A goal of the DA was to be very simple, performing only set management routines. As such, only five new methods are defined. Table 7 highlights those methods, and they are explained in detail in Appendix C.

TABLE 7. DA API.

Method	Description
<code>da_put</code>	insert a data object into the archive
<code>da_delete</code>	remove a data object from the archive
<code>da_list</code>	display the holdings of the archive
<code>da_info</code>	display metadata about the archive
<code>da_get</code>	redirects to the object's URL or URN

The DA does not disable any of the currently defined bucket methods. Some of the methods may be unnecessary, but they were left in they were left in for completeness. For example, end users are not meant to interact directly with DAs; DAs exist to aid in the construction of DLs. However, the “display” method was left in the DA because: 1) a user might “stumble” across a DA, and it should be able to generate a human readable display; and 2) an archive might have need to store human consumable information in regular packages and elements – for example links to all the DLs that harvest from the archive. It might be advantageous for DA's to have their standard methods overridden with implementations tailored to archive application. However, a DA's main traffic is expected to remain DLs calling the various `da_*` methods.

4.2.2 Changes From a Regular Bucket

The five `da_*` methods are stored as regular methods in the standard `_methods.pkg` package. However, DAs also have a DA-specific package, `holdings.pkg`, which contains library source files as well as the databases generated to store the objects in the DA. A tool for duplicating a DA's holdings could simply retrieve

(modulo the correct T&C) the known elements from this package to get a “copy” of the DA’s contents.

Similarly, a regular bucket could be changed into a DA through the “pack” / “unpack” methods to extract and replicate the contents of the `holdings.pkg` package and the five DA methods. Furthermore, if desirable for a specific application, a bucket could serve “double duty” – responding to `da_*` methods from DLs, and all the while serving “regular” data contents to users interacting with the bucket through the normal bucket methods.

4.3 Discussion

Even for the limited goals of a dumb archive, the current implementation only scratches the surface of the work that could be done. In this section, some of the systems issues of DA implementation are discussed, and an outline is given on how other archive protocols could be implemented using DA.

4.3.1 DA Examples

Although interaction with a DA should occur through a software tool interface, we can examine the methods used to populate and interact with the DA. Consider an installed DA:

```
http://dlib.cs.odu.edu/da/
```

This will appear as a regular bucket to someone that goes directly to the above URL. If it is known that the URL is a DA, then the items registered with the archive can be listed with:

```
http://dlib.cs.odu.edu/da/?method=da_list
```

The above URL will produce a list of ids and URLs for all items registered with the archive, or a null list if nothing is registered. Items can be placed in the archive:

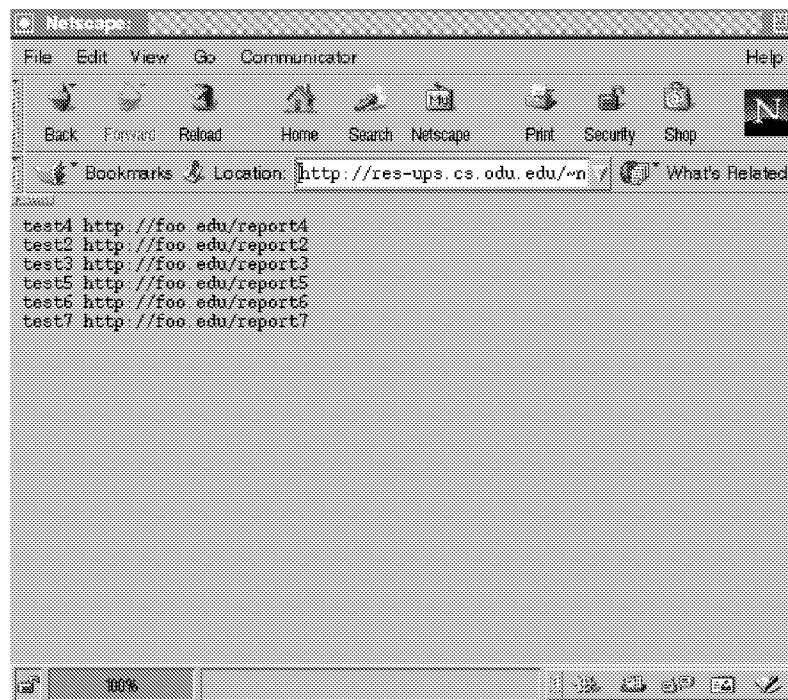


FIG. 15. DA query (?method=da_put&adate=<20000101).

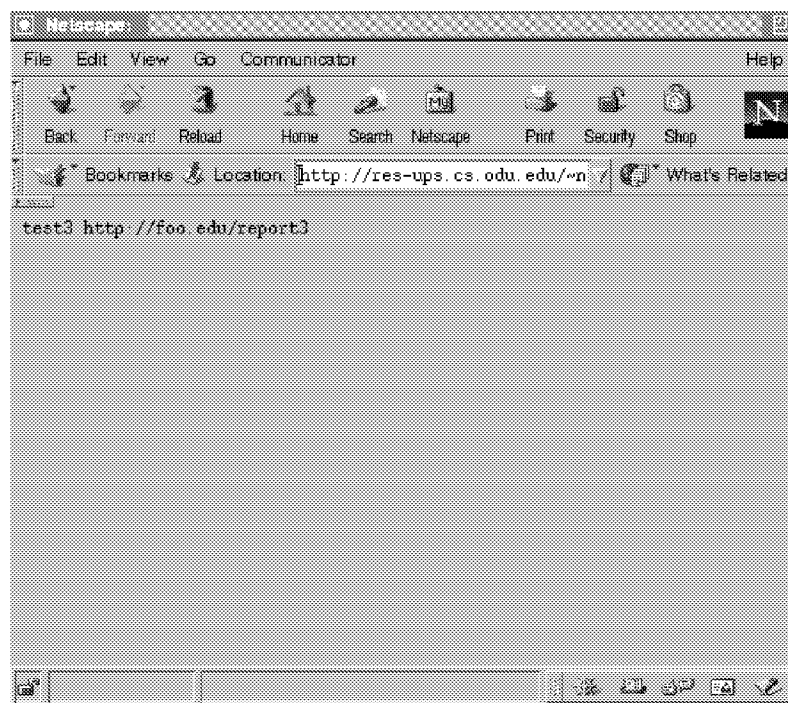


FIG. 16. DA query (?method=da_put&adate=19940101-20000101&subject=cs).

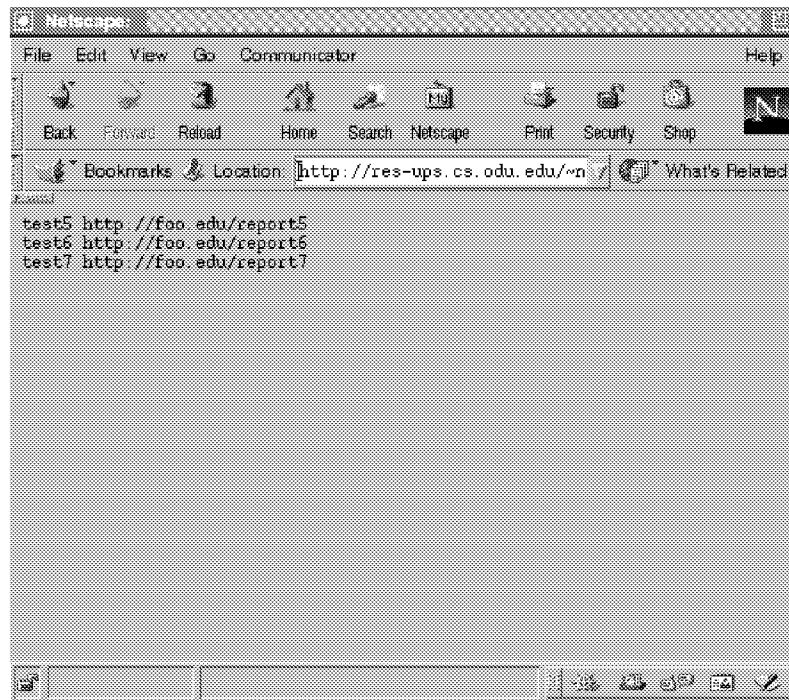


FIG. 17. DA query (?method=da_put&subject=phys).

4.3.2 DBM Implementation Notes

At first glance, it is tempting to implement the DA functionality using the package and element constructs of the bucket. Packages could be defined for each of “da_put” arguments, and an element with record’s id in those packages would contain the values for those arguments. However, as discovered during the UPS implementation (detailed in Chapter Six), the Solaris operating system will allow only 32,767 inodes within a single directory (Sun Microsystems, 1999). The current bucket implementation would have scalability difficulties with archives containing more than 32,767 records. To alleviate this problem, a different internal data structure was used for the DA functionality instead of the package / element semantics.

The internal data structures used by DA are implemented with variants of the Berkeley Database Management (DBM) library (Olson, Bostic & Seltzer, 1999). An

index is built for each the possible arguments to “da_put”: “id”, “url”, “adate”, “pdate”, “subject”, and “metadata” (see Appendix C for a detailed description). DA attempts to use the Gnu version, GDBM, but this library is not always available on standard Unix distributions. If GDBM is not available, it will use the NDBM version, which is available on all systems.

GDBM is preferable to NDBM (and the other standard versions, SDBM and ODBM) because the GDBM does not have the limitation of the others of the key + hash for an entry having a total size of 1024 bytes. As such, if GDBM is not available on the system running DA, typical values for “metadata” will exceed 1024 bytes and the “da_put” will not be successful.

The DBM libraries provide a convenient and lightweight database mechanism for the DA, but it does come at a cost to the bucket. Whereas “normal” buckets are mobile and can move from server to server, a DA has mobility only within homogeneous architectures. DBM files are binary and differ for various machine architectures. Furthermore, DBM variants are not interchangeable, so if a DA began with GDBM, it could not automatically read the data files using NDBM. However, given the permanent nature of archives, for most applications, non-mobile archives will not be a problem. Not only is the DA an example of a modified bucket, but it is also an example of how specialization impacts the general bucket requirements.

4.3.3 Open Archives Initiative Dienst Subset Mapping

The currently evolving Open Archives initiative (OAi) aims at making the technology available for information providers to open up their archives for digital library service providers to harvest their contents, apply their value-added processing, and present them to their targeted customer base (Van de Sompel & Lagoze, 2000). The OAi is a DODA DL model, with sophisticated user services expected to be built from the harvesting of multiple OAi-compliant archives. While previous attempts within the DL community at defining common functionality for archives have generated “limited consensus” (Scherlis, 1996), the OAi has bounded archive expectations by focusing on

what is achievable in the short term. The OAi has defined a small subset of the popular Dienst protocol, known as the “Open Archives Dienst Subset” (Davis, Fielding, Lagoze, & Marisa, 2000), which has the sole purpose of aiding service providers in harvesting archives.

The OAi is an evolving protocol, with version 2 expected in December 2000. Once the OAi protocol has stabilized, it is anticipated that it will be easy to implement this protocol through a mapping into DA commands. A site using an OAi enabled DA will be able to respond to generic OAi harvesting requests, but also have the additional capabilities of the DA, such as T&C – which are currently not part of the OAi protocol, or return metadata in non-extensible markup language (XML) encodings. Table 8 shows the currently defined OAi Dienst subset “verbs” and their DA equivalents.

Some of the concepts do not directly map because the OAi Dienst heritage emphasizes the preeminence of archives and DLs, where buckets emphasize the objects themselves as the canonical source. For example, DAs can provide the metadata via “Disseminate” as a convenience to the harvester, however the bucket remains the canonical source of metadata about itself. Similarly, if the harvester wants the metadata in a certain format, the expected procedure relative to buckets is to ask the bucket itself, not the bucket’s archive. However, the DA could be modified to perform these services on the harvester’s behalf since not all harvesters will be bucket-aware.

Another area of concern is that Dienst encodes its messages in non-standard CGI format, requiring modification of the http server configuration to successfully trap the incoming http requests and route them to the correct location. In comparison, all bucket messages (including DA messages) are entirely encapsulated in the http message and require no server modifications. While not difficult to develop, a script would be required to trap the incoming Dienst messages and re-route them into the DA format.

TABLE 8. OAi → DA mapping.

OAi Dienst	DA	Discussion
Subset Verb	Method	
Disseminate	da_list	“da_list” with the “id” (or “url”) and “metadata” argument currently fulfills the same purpose of the “Disseminate” verb. “Disseminate” does support the possibility of retrieving different metadata formats, while “da_list” only returns what was originally uploaded. “da_list” could be modified to perform this service as a convenience to the harvester (calling the bucket’s “metadata” method on the harvester’s behalf).
List-Contents	da_list	“da_list” as currently implemented is not as general as “List-Contents”, but the “da_list” arguments “adate” and “subject” provide the same functionality. If other partitions (or “clusters”) for the DA are defined, they could be included in the same manner as “subject”. Note that “List-Contents” does not by default support the concept expressed in the “pdate” argument to “da_list”.
List-Meta- Formats	da_info	DAs could list their native metadata format(s) as part of the “da_info” method.
List-Partitions	da_info	DAs could either list the partitions supported as part of the “da_info” method, or it could just the list the pre-defined partitions (or “clusters”) of the DA.
Structure	n/a	“da_list” could be modified to provide this capability, but metadata conversion is really the provenance of the bucket (assisted through the BCS).

CHAPTER FIVE

BUCKET COMMUNICATION SPACE

5.1 Overview

The Bucket Communication Space (BCS) is partially motivated by Linda, the parallel communication library (Carriero & Gelernter, 1989). In Linda, processes effectively pass messages by creating “tuples” that exist in “tuple space”. These data objects are created with the “eval” primitive, and filled with data by processes using the “out” primitive. Processes use “rd” and “in” for reading and reading-removing operations, respectively. These primitives allow processes to communicate through tuple space, without having to know the details (e.g. hostnames, port numbers) of where the processes are. The messages written to tuple space can have regular expressions and control logic to specify who should read them. When a “in” tuple sees an “out” tuple and the conditions of the former match that of the latter, the message is communicated to the receiving process and the tuple is removed from tuple space. Though it imposes a performance overhead, the Linda environment provides a useful layer of abstraction for inter-process communication.

We wished to provide something similar for buckets: buckets communicating with other buckets without having to know the details of bucket location. This is especially important if the buckets are mobile, and a bucket’s location is not guaranteed to be static. The BCS also provides a method for centralizing functionality that cannot be replicated in individual buckets. This could be either because of efficiency concerns (the resulting bucket would be too bloated) or implementation limitations (a service is not available on the architecture that is serving the bucket). Buckets need only know how to communicate to a BCS server, which can handle their requests for them.

Buckets maintain the location of their BCS server through a bucket preference. This allows for the specification of a single BCS server, with no provisions for if that

BCS server is not available. Currently, no detailed plans have been made for complex BCS architectures. There is no built-in concept of a master BCS server for all buckets, localized BCS servers, rings of BCS servers or any other architectural possibilities. If these architectures are to be built, it will involve the modification of the BCS buckets to recognize peer BCS buckets, master BCS buckets, etc. However, these modifications should be transparent to the data buckets themselves, with data buckets still only tracking the location of their entry into the bucket communication space.

The BCS model opens up many possible service areas. A subtle element of the BCS is that buckets, not people, are responsible for the provision and coordination of these services. We provide proof-of-concept implementations for four significant services: file format conversion, metadata conversion, bucket messaging, and bucket matching.

5.1.1 File Format Conversion

File format conversion provides bi-directional conversion of image (e.g. GIF, JPEG) formats and page description formats (e.g., PostScript, PDF). Format conversion is an obvious application – additional formats will become available after a bucket’s publication and the ability to either place them in the bucket or dynamically create them will be useful in information migration.

5.1.2 Metadata Conversion

Metadata conversion is similar to file format conversion, providing conversion between some of the more popular metadata formats (e.g., Refer, RFC-1807, bibtex). Metadata conversion is extremely important because although buckets ultimately have to choose a single format to operate on, it is unreasonable to assume that all applications needing metadata from the bucket should have to choose the same format. Being able to specify the desired format to receive from a bucket also leaves the bucket free to change its canonical format in the future.

5.1.3 Bucket Messaging

Messaging allows multiple buckets to receive a message if they match specific criteria. While point-to-point communication between buckets is always possible, bucket messaging provides a method for discovering and then sending messages to buckets. Messaging provides functionality closer to the original inspiration of Linda, and can be used as the core of a “bucket-multicasting” service that sends pre-defined messages to a subset of registered buckets. This could be used in turn to implement a metadata normalization and correction service, such as that described by French, Powell, Schumann, & Pfaltz (1997) or Lawrence, Bollacker, & Giles (1999).

5.1.4 Bucket Matching

The most compelling demonstration of the BCS is bucket matching. Matching provides the capability to create linkages between “similar” buckets. Consider a technical report published by the Old Dominion University computer science department that is also submitted to a conference. The report exists on the DL maintained by the department and the publishing authority is: `ncstrl.odu_cs`. If the conference paper is accepted, it will eventually be published by the conference sponsor. For example, say the conference sponsor is the Association for Computing Machinery, whose publishing authority would be `ncstrl.acm`. Although the conference paper will surely appear in a modified format (edited and perhaps abbreviated), the technical report and the conference paper are clearly related, despite being separated by publishing authority, date of publication, and editorial revisions. Two separate but related objects now exist, and are likely to continue to exist.

How best to create the desired linkage between the two objects? It is easy to assume `ncstrl.acm` has neither the resources nor the interest to spend the time searching for previous versions of a manuscript. Similarly, `ncstrl.odu_cs` cannot link to the conference bucket at the creation time of the technical report bucket, since the conference bucket did not exist then. It is unrealistic to suggest the relevant parties will go back to the `ncstrl.odu_cs` archive and create linkages to the `ncstrl.acm` bucket after six months to a

year have passed. However, if both buckets are registered in the same bucket communication space (by way of sending their metadata or fulltext), they can “find each other” without human intervention. When a match, or near match (the threshold for “match” being a configurable parameter) is found, the buckets can either automatically link to each other, or inform a human reviewer that a potential match has been found and request approval for the linkage.

This technique could also be used to find related work from different authors and even duplications (accidental or plagiarious). In the test runs using the NACA portion of the Universal Preprint Service (see chapter six), find multi-part reports were found (e.g. Part 1, Part 2), Technical Notes (archival equivalent of a computer science technical report) that were eventually published as Reports (archival equivalent of a journal article), and a handful of errors where duplicate metadata was erroneously associated with multiple reports.

5.2 Implementation

While the current BCS implementation lacks the elegance of Linda, it is easy to implement. Similar to DA, instead of developing an entirely new application for the BCS, buckets were modified to have BCS-specific methods. Also similar to the DA, none of the standard bucket methods were removed in the BCS, even though it is not envisioned that end users working directly with the BCS. Although presented as two separate buckets, it is possible for a single bucket to be both a BCS server and a dumb archive. However, BCS differs from DA in that the DA makes no assumption that the objects in the DA are buckets, but the BCS does assume that all of the objects it has registered are in fact buckets.

5.2.1 Implemented Methods

Table 9 includes a short summary of the BCS methods, and Appendix D covers them in detail. “bcs_register”, “bcs_unregister”, and “bcs_list” are used to manage the internal data structures for inclusion in the BCS. The BCS uses the DBM variants, GDBM or NDBM, for its internal storage just as the DA does.

“bcs_convert_image” is simply a wrapper to the Image Alchemy conversion program (Image Alchemy, 2000). Any conversion program could be used, such as the popular freeware product ImageMagick (ImageMagick, 2000). In fact, it would have been preferable to use ImageMagick, not only because it was free but also because it includes a Perl module for easy conversion and manipulation from inside a script. However, ImageMagick was not installed on the development machines, so Image Alchemy was used instead. It would also be possible to implement “bcs_convert_image” using a suite of tools instead of just one, or to implement a more sophisticated format conversion environment, such as the Typed Object Model (TOM) Conversion Service (Ockerbloom, 1998). Although Image Alchemy supports over 100 image formats, the current version of “bcs_convert_image” only implements the popular TIFF, GIF, JPEG, PNG, PostScript and PDF formats for demonstration purposes.

TABLE 9. BCS API.

Method	Description
bcs_convert_image	converts an uploaded image to a specified format
bcs_convert_metadata	converts an uploaded metadata file to another metadata file format
bcs_list	lists all the buckets registered with the BCS
bcs_match	finds & creates linkages between all “similar” buckets
bcs_message	identifies buckets that match a specific criteria, and sends them a message
bcs_register	registers the bucket into the BCS
bcs_unregister	unregisters the bucket from the BCS

“bcs_convert_metadata” is a wrapper for our own metadata translation program, mdt (Nelson, et al, 1999). In the course of implementing various DL projects, a host of metadata translation scripts have been developed – some generalized, some highly

specialized. Furthermore, there are a number of other metadata translation programs freely available, such as “bp” (Jacobsen, 1996) and “InterBib” (Paepcke, 1997). Many of these programs have overlapping format coverage and none perform all conversions with equal proficiency. Ideally, “bcs_convert_metadata” should be constructed from the union of the best metadata conversion programs, not just a single one. However, for demonstration, only mdt is used and the following formats are supported: refer (Lesk, 1978), bibtex (Knuth, 1986), RFC-1807, Dublin Core (Weibel, Kunze, Lagoze, & Wolfe, 1999), and the Open Archives Metadata Set (OAMS) (Van de Sompel & Lagoze, 2000).

“bcs_message” searches through all the registered buckets, looking for those that match a regular expression passed in as an argument. “bcs_message” can either return the unique ids / URLs of the matching buckets, and/or send the matching buckets a message (also passed in as an argument).

“bcs_match” searches through all the registered buckets, either comparing all of them against all of them, or a list (passed in as an argument) of buckets against all of them. “bcs_match” considers only the metadata passed in during registration when computing similarity. To determine similarity, “bcs_match” uses the cosine correlation with frequency term weighting (CCFTW), first used by Salton & Lesk (1968). Adapting Harman’s (1992) definition of CCFTW to document-document comparison (instead of document-query), similarity is defined as:

$$\text{similarity}(d_j, d_k) = \frac{\sum_{i=1}^n (td_{ij} \cdot td_{ik})}{\sqrt{\sum_{i=1}^n td_{ij}^2 \cdot \sum_{i=1}^n td_{ik}^2}}$$

where

td_{ij} = the i^{th} term in the vector for document j

td_{ik} = the i^{th} term in the vector for document k

n = the number of unique terms in documents j and k

The CCFTW returns a number between 0 and 1. For the testbed of 3036 NACA documents, it was informally determined that a useful threshold for similarity was 0.85. Numbers much below 0.85 did not appear similar on inspection. Conversely, numbers at or above 0.93 were almost always the “same” document published in another version (i.e., NACA TN vs. NACA Report). The similarity threshold is tunable parameter – different corpora may require different thresholds.

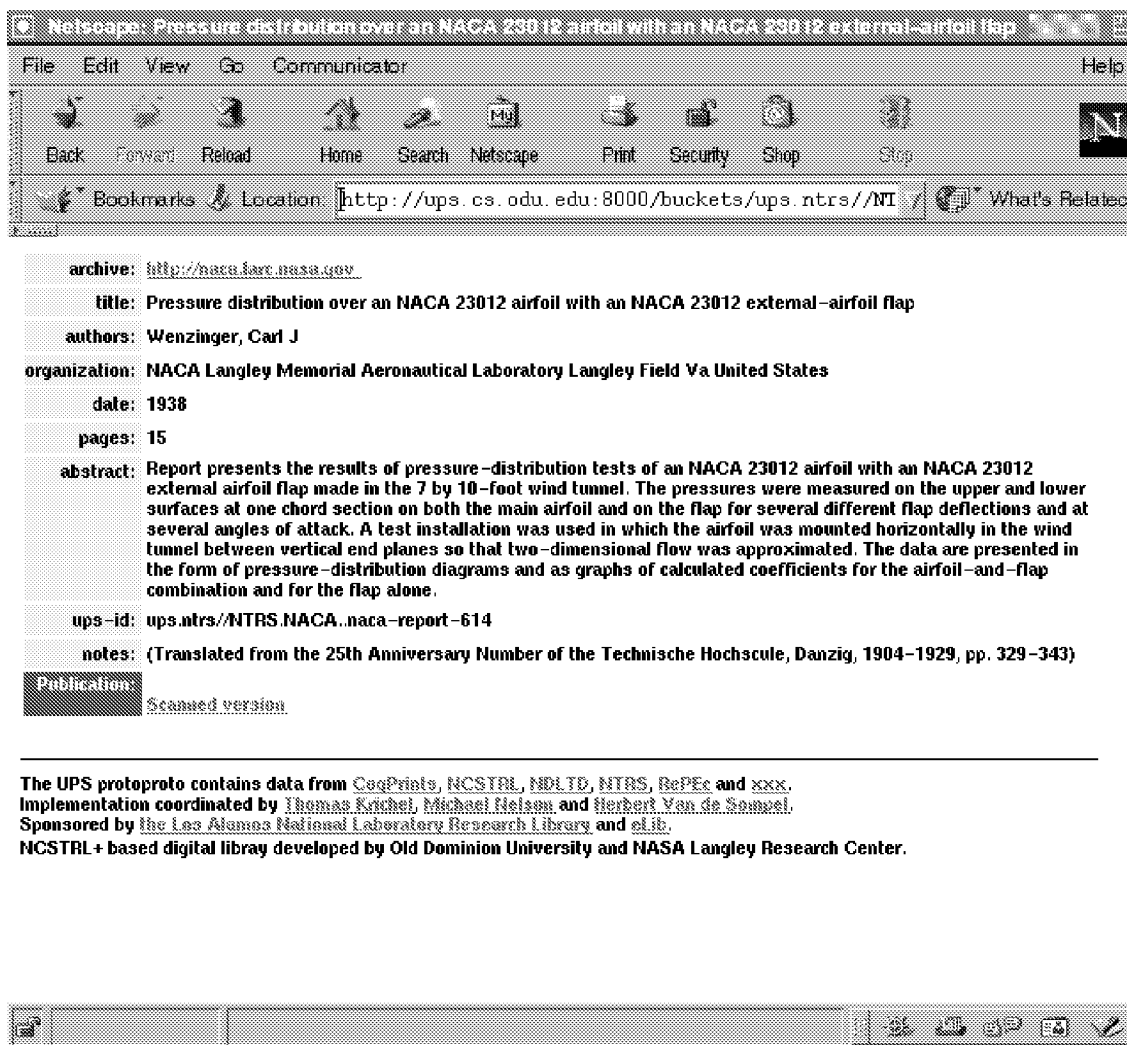


FIG. 18. NACA bucket before similarity matching.

To initiate similarity matching for the contents registered with the BCS, the following message would be sent:

```
http://dlib.cs.odu.edu/bcs/?method=bcs_match
&threshold=0.90&report=on&link=on
```

The “threshold” argument resets the definition of relevancy for the value returned from the CCFTW function. The “report” argument stores the results of the matching in the BCS bucket for later perusal (the default value is to discard the results), and the “link” argument instructs the BCS server to attempt the buckets that are found to be similar (the default action is to report the findings but not link). An example of a NACA report before and after similarity matching is shown in figures 18 and 19.

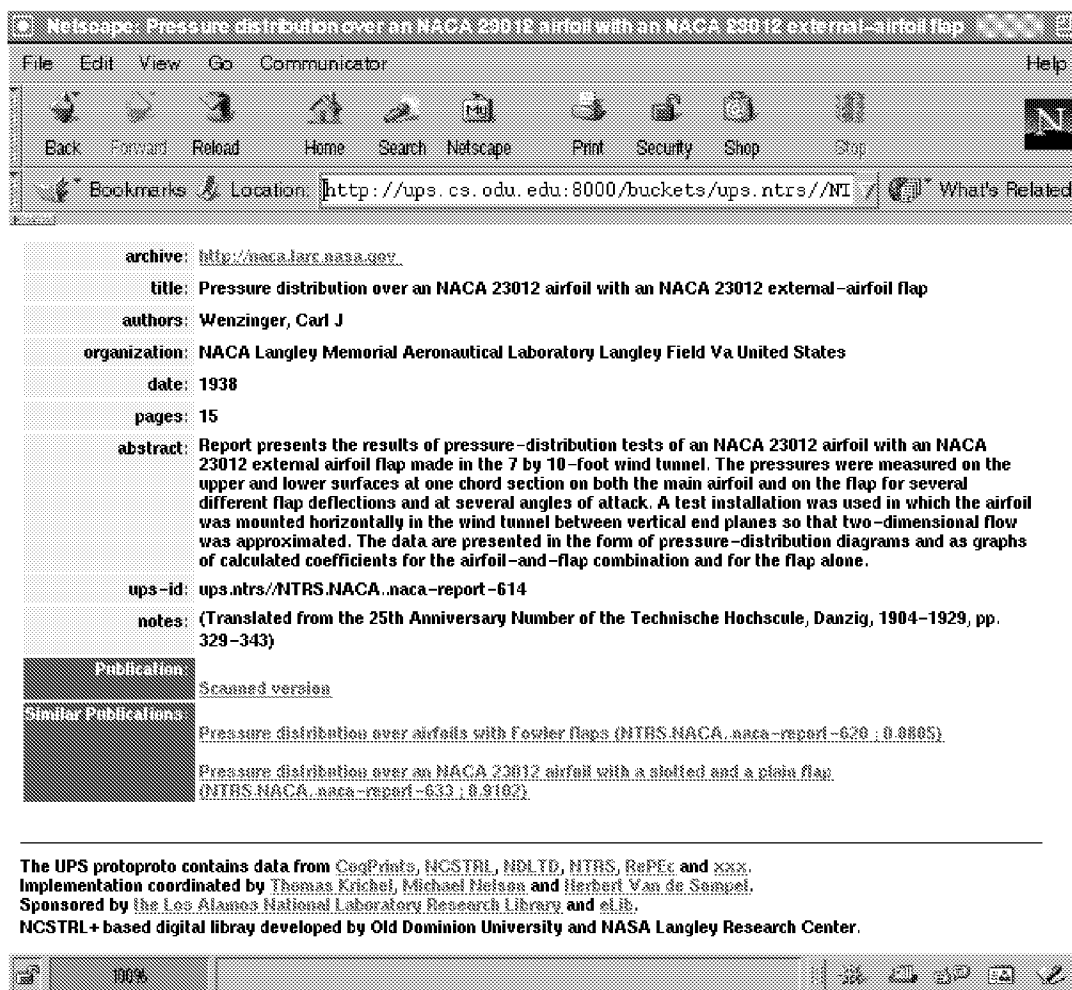


FIG. 19. NACA bucket after similarity matching.

Other similarity measures are possible, including the inverted document frequency (IDF) measure (Sparck Jones, 1972) and its variations (Sparck Jones, 1979; Croft & Harper, 1979), and the 2-Poisson model (Bookstein & Swanson, 1974). Also, it could be possible to implement the algorithms used in duplication and plagiarism detection systems such as SCAM (Shivakumar & Garcia-Molina, 1995), MDR (Monostori, Zaslavsky, & Schmidt, 2000), and dup (Baker, 1995a). However, the purpose of “bcs_match” was not to test which measures are best, but rather to simply prove the BCS could perform the service.

If a DL is used to discover a bucket with BCS similarity links, the search results page of the DL will likely have many of the same buckets listed there that are listed as similarity links in the bucket. However, depending on the search criteria (such as searching on authors or dates instead of abstract keywords), the search results and similarity links could also differ significantly. The similarity links provide a fixed navigation structure for the corpus that does not change as search criteria change. Also, the presence of BCS similarity links does not preclude the existence of a complimentary value-added service that performs dynamic searches into other DLs for documents similar to the current bucket. A dynamic similarity service would offer more flexibility in finding relevant documents, but it also assumes the continued existence and accessibility of the service.

5.2.2 Changes From a Regular Bucket

A BCS bucket is similar to a regular bucket, but with the seven new `bcs_*` methods in the `_methods.pkg` package. Also, a new package, `bcs.pkg`, is added to contain all the support libraries, programs and data files for BCS operation.

The resulting BCS bucket is larger and even less mobile than the DA. A BCS bucket carries two of its support programs with it: `mdt` and Image Alchemy. `mdt` is a Perl program, so it is portable, but Image Alchemy is binary program (approximately 3MB) that is obviously not portable between architectures. BCS buckets also use DBM

variations for their registration data structures, and thus inherit their portability limitations. However, it is unlikely that BCS buckets will be expected to be mobile, and since a site will probably not have more than one BCS bucket (or at most a few), their increased size should not be a problem.

5.3 Discussion

It should be stressed that the newer BCS buckets lag behind regular buckets in their development and maturity, regular buckets having the benefit of several years of testing in production environments. Combined with the fact that BCS buckets perform more sophisticated tasks, a review of BCS operation should be considered proof-of-concept of BCS operation, and not the final representation of their capability and performance profile.

5.3.1 Performance Considerations

Two of the BCS methods have significant time requirements for their operation, and are the first targets for optimization. “bcs_message” performs a linear search through all the registered buckets searching for those that match the requested regular expression. If the BCS bucket used an existing search engine, or implemented its own inverted files for this purpose, “bcs_message” would run in $O(\log n)$ time.

Even more inefficient is the “bcs_match” method, which currently runs in $O(n^2)$. This is because the default case is to compare everything to everything else. With the 3036 NACA documents in UPS as the testbed, the similarity matching was run on all of the documents. 3036 documents require 9,217,296 comparisons. However, since similarity is bi-directional, only half that many were computed. Similarity matching for a corpus can be thought of as filling a matrix similar to Figure 20. The diagonal is all 1’s (since documents are always completely similar to themselves), and the bottom half of the matrix is simply a duplication of the top half.

The implementation eventually optimized to the point where it could complete approximately 576,000 comparisons per hour while running on the NACA collection, finishing the NACA documents in approximately 8 hours. The largest collection

“bcs_match” has actually been tested on is 6,867 documents (UPS NACA (3036) + UPS Math (3831)). Similarity matching on this collection ran in approximately 42 hours, for approximately 561,000 comparisons per hour. The final results found no similar documents between the two collections, and 159 matches in the NACA collection and 35 matches in the Math collection.

Computing similarity is hard. For example, even though Dienst provides a user option for ranking the search results by relevancy, it remains unimplemented. The commonly available Wide Area Information Server (WAIS) search engine (Kahle, Morris, Davis, Tiene, Hart, & Palmer, 1992) implementations arbitrarily limit the set of returned documents to be in the range of 200 - 450, which allows similarity computation to be tractable.

	id-1	id-2	id-3	id-4	...	id-n
id-1	1	0.298	0.783	0.267	...	0.459
id-2		1	0.976	0.732	...	0.432
id-3			1	0.868	...	0.291
id-4				1	...	0.870
...					1	0.904
id-n						1

*not computed -
same as above
the diagonal*

FIG. 20. Sample similarity matching matrix.

An obvious optimization would be to use inverted files, and perform similarity matching on only those documents that have a minimum level of intersection between them. This is similar to what a search engine does: finding the documents that have the same keywords as the query, and performing similarity matching only on those documents. However, “bcs_match” would not gain the same great reduction in the search space because typical queries are only a few words, so filtering through an inverted file is likely to produce only a small number of documents. In “bcs_match”, the query is actually an entire document, so the search space would not be reduced the same amount as a relatively small query.

Another optimization, somewhat related to the above, is to use a clustering technique (Rasmussen, 1992) to partition the corpus into a smaller number of “related” sections, and perform similarity matching only within those partitions. This is illustrated in the similarity run with the combined NACA and Math collections. That run took 42 hours, and found no matches between the two collections. When the similarity matching is run on the two collections sequentially, both runs can be completed in approximately 8 hours each.

Both clustering and inverted files address the similarity matching problem by reducing the search space, clearly a necessary optimization for an $O(n^2)$ algorithm. However, clustering merely postpones the problem. If the size of the collection grows to 30,000,000 documents, and clustering techniques are employed to produce partitions of, for example, 30,000 documents, then similarity matching will still require $O(n^2)$ within that cluster. The problem could be further postponed if more efficient implementations of “bcs_match” could yield dramatic improvements over 570,000 comparisons per hour.

Another optimization approach would be to exploit the parallelizable nature of similarity matching. Consider partitioning the similarity matrix such that regions of the matrix were assigned to separate computers (Fig. 21). No communication between computers handling different regions is necessary; they could simply report their results back to the BCS server that would then collate their results. This could be accomplished

by harvesting idle workstation cycles (Kaplan & Nelson, 1994; Baker, 1995b), or even through a specialized screen saver similar to the popular SETI@Home, which taps the power of idle personal computers (Sullivan, Werthimer, Bowyer, Cobb, Gedy, & Anderson, 1997). This approach to similarity matching could be pursued independently of other possible optimizations.

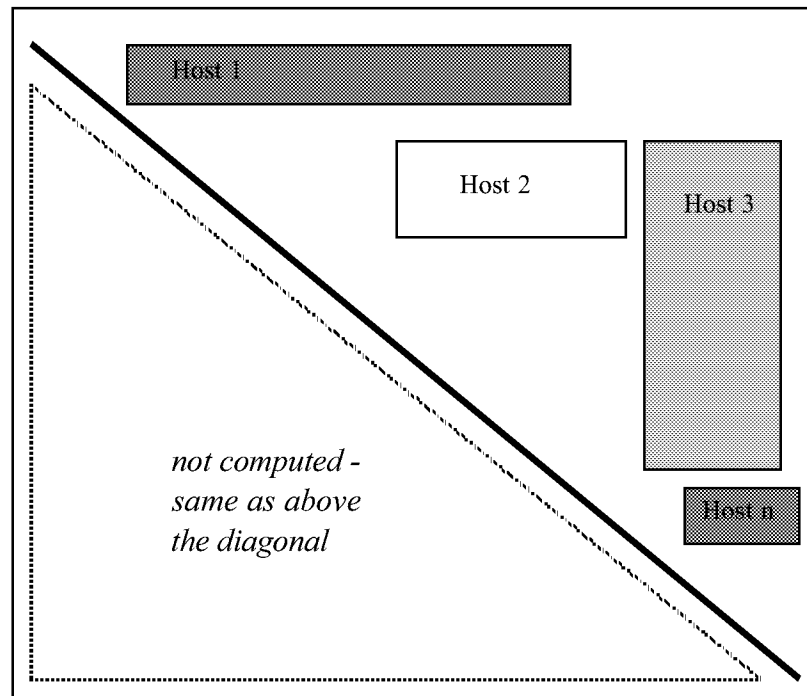


FIG. 21. Partitioning of the similarity matching matrix

5.3.2 Current Limitations

The BCS puts a solid foundation in place, but as yet individual buckets and the BCS buckets themselves have not tapped its real potential. The BCS does not yet have the “killer app” needed to unequivocally demonstrate its usefulness. The similarity matching is a good candidate, but the performance limitations of current implementation make it less than compelling. Designing and implementing the “perfect” similarity matching solution is a significant undertaking in its own right. We hope the easily extensible nature of the BCS buckets will encourage others to optimize existing, or develop new BCS services.

Another limitation is the messages passed through the BCS (or even bucket-to-bucket) are defined in terms of the already existing bucket API. That is, there is currently no way to construct a message for a bucket to specify something other than what is available through the bucket API. An unexplored realm would be defining general bucket messages, perhaps encoded in the Knowledge Query Manipulation Language (KQML) (Finin, Fritzson, McKay, & McEntire, 1994), to specify a bucket's beliefs, desires and intentions. In short, buckets are intelligent, but not as intelligent as they can be, and the BCS is the infrastructure that will facilitate the intelligence growth of buckets.

CHAPTER SIX

BUCKET TESTBEDS

6.1 NCSTRL+

Bucket development was begun within the context of the NCSTRL+ project. NCSTRL+ is the result of several years of research and development in digital libraries (Fig. 22). In 1992, the ARPA-funded CS-TR project began (Kahn, 1995) as did LTRS. In 1993, WATERS (Maly, French, Fox, & Selman, 1995) shared a code base with LTRS. In 1994, LTRS launched the NTRS, and the CS-TR and WATERS projects formed the basis for the current NCSTRL (Davis & Lagoze, 2000). In 1997, NCSTRL+ was begun, drawing from the contents of NCSTRL and NTRS.

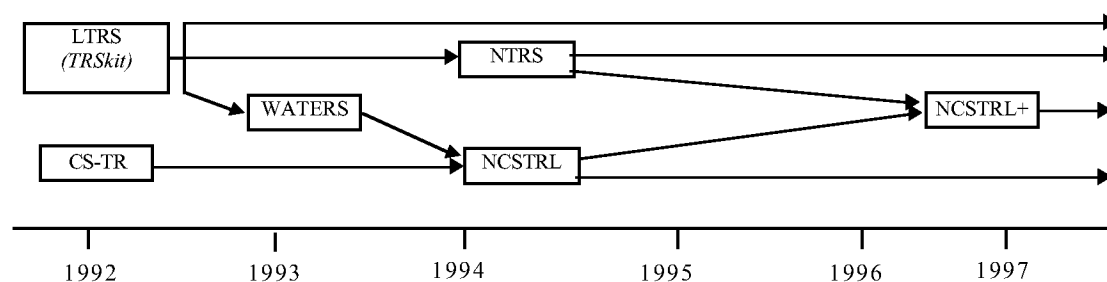


FIG. 22. NCSTRL+ lineage.

NCSTRL+ was used as the primary testbed for buckets and other DL technologies from 1997-1999. NCSTRL+ contains approximately 1000 buckets drawn from LTRS as well as a handful of buckets constructed for various Old Dominion University research projects. NCSTRL+ also provides distributed searching into the NCSTRL collection.

6.1.1 Dienst

Dienst was chosen to implement NCSTRL+ instead of other digital library protocols such as TRSkitt (Nelson & Esler, 1997) because of Dienst's success over several years of production in NCSTRL. Dienst appeared to be the most scalable, flexible, and

extensible of digital library systems surveyed (Esler & Nelson, 1998), but scalability limitations were discovered in the reference implementation (Van de Sompel, Krichel, Nelson, et al., 2000b). However, this was a limit of the reference implementation, not the protocol itself. Dienst has also been used in a variety of DL experiments, such as the Networked Digital Library of Theses and Dissertations (Fox, Eaton, McMillan, et al., 1997), the Electronic Library for Grey Literature (part of the MeDoc project) (Adler, Berger, Bruggemann-Klein, et al., 1998), the European Technical Reference Library (Andreoni, Bruna Baldacci, Biagioni, et al., 1998), the ACM-sponsored Computing Research Repository (CoRR) (Halpern & Lagoze, 1999), and most recently the Open Archives initiative data harvesting protocol (Van de Sompel & Lagoze, 2000). For NCSTRL+, Dienst 4.1.8 is used. Dienst 5.0 has some significant architectural changes that make it less suitable for these particular research purposes. Most notable is the switch from a distributed search to a centralized search mechanism. Full distributed searching was abandoned for the production version of NCSTRL because of the low availability of the entire distributed system. One study from 1997-1999 found that at least one of the nodes in NCSTRL was always unavailable (Powell & French, 2000).

While Dienst is discipline independent, it is currently discipline monolithic. It makes no provision for knowledge of multiple subjects within its system. While it is possible to set up a collection of Dienst servers independent of NCSTRL, there is no provision for linking such collections of servers into a higher level meta-library. A collection service has been proposed that would allow for partitioning of a server's holdings (Lagoze & Fielding, 1998), but the collection service is not in production use.

Dienst consists of 5 components: 1) Repository Service; 2) Index Service; 3) Meta-Service; 4) User Interface Service; and 5) Library Management Service. Each of the services has a list of valid "verbs" that the service understands, and some of the verbs can take arguments. Dienst uses http as a transport protocol. The standard format is:

```
http://machine.name:port/Dienst/Service/Version/  
Verb/Arguments
```


An example of a valid Dienst request is:

```
http://repository.larc.nasa.gov:8080/Dienst/Meta/
2.0/Publishers/
```

This contacts the Meta-Server service at repository.larc.nasa.gov and requests a list of publishing authorities that this machine contains. Dienst names objects in collections using handles, a URN implementation from the Corporation for National Research Initiatives (CNRI). NCSTRL+ uses the experimental and unregistered handles of “ncstrlplus.larc” and “ncstrlplus.odu.cs”. Meta-data for objects is stored in RFC-1807 format.

The basic architecture of NCSTRL has a single entry point (“home page”) for user access. Each publishing authority (in practice, an authority generally corresponds to a university department or laboratory) runs its own copy of the Dienst software. The home page gathers the queries and dispenses the queries in parallel to each server, gathers the results, and displays the correlated results to the user. To assist with performance and reliability, Dienst now employs a Regional Meta-Server (RMS) to partition all NCSTRL participants into geographic regions. The various RMSs share their data with the Master Meta Server (MSS) at Cornell (the home of Dienst and NCSTRL). A Merged Index Server (MIS) provides a single index of all the metadata outside a region. A search query is sent to all standard sites within a region, and to the region’s MIS for metadata outside the region.

6.1.2 Clusters

While Dienst is a successful production quality DL protocol, it has some inherent limitations that prevent additional features from being added. Among these is the inability to subdivide collections along anything other than institutional boundaries.

Clusters are a way of aggregating logically grouped sub-collections in a DL along some criteria. NCSTRL+ provides 4 clusters: organization, archival type, terms and

conditions, and subject category. Organization is the “publishing authority” that is included by default in Dienst. Archival type includes specifies the semantic type, such as pre-print, technical report, software, datasets, etc. The terms and conditions cluster specifies the access restrictions associated with the object, such as free, password or monetary charge required, etc. For subject category, the NASA STI categories were adopted and modified slightly to be less aerospace-centric (Tiffany & Nelson, 1998). A two-level hierarchy, there are 11 top-level categories, and each category has approximately 10 sub-level categories. This provides subject classification that is broad and lightweight.

6.2 Universal Preprint Service

The Universal Preprint Service (UPS), which has since been renamed the Open Archives initiative (OAi), is a much larger DL testbed introduced in October 1999 and is based on NCSTRL+ software. The UPS prototype was a feasibility study for the creation of cross-archive end-user services. With the premise that users would prefer to have access to a federation of digital libraries, the main aim of the project was the identification of the key issues in actually creating an experimental end-user service for data originating from important existing, production archives. This included a total of almost 200,000 buckets harvested from six existing production DLs. Table 10 provides a list of the archives and their contributed content. A full discussion of the results from the UPS project can be found elsewhere (Van de Sompel, Krichel, Nelson, et al., 2000a; Van de Sompel, Krichel, Nelson, et al., 2000b). The two key bucket-related technologies are lightweight buckets and SFX reference linking within buckets.

TABLE 10. UPS participants.

Archive / DL	Records in DL	Buckets in UPS	Buckets Linked to Full Content
arXiv	128943	85204	85204
www.arxiv.org			
CogPrints	743	742	659
cogprints.soton.ac.uk			
NACA	3036	3036	3036
naca.larc.nasa.gov			
NCSTRL	29680	25184	9084
www.ncstrl.org			
NDLTD	1590	1590	951
www.ndltd.org			
RePEc	71359	71359	13582
netec.mcc.ac.uk			
Totals:	235361	187115	112516

6.2.1 Lightweight Buckets

Only the metadata was harvested from the six archives – not the actual content itself. While harvesting the full content would have been technically possible, it would have been storage intensive and would have added little to the cross-archive demonstration. The resulting buckets were dubbed “lightweight buckets”, since they contained only the metadata and pointers back to the content in the original archives. However, the lightweight buckets proved to be useful containers for additional material and value added services that could not be added to the original archive. Although the BCS was not completed at the time of the UPS demonstration, the lightweight buckets have since served as mount points for BCS value-added services, such as bucket matching. There is an entire class of applications where it is desirable to aggregate information about

an object, but the original object cannot be moved due to storage constraints or intellectual property restrictions. For UPS, metadata was aggregated in multiple formats, (RFC-1807 and ReDIF (Cruz & Krichel, 1999)), and used the buckets as attachments for services such as the SFX reference linking service.

6.2.2 SFX Reference Linking in Buckets

The SFX reference linking service (Van de Sompel & Hochstenbach, 1999) is a dynamic layer of abstraction between bibliographic information objects and potential library databases and services. The traditional library has an array of commercial databases and services, such as:

- ISI's Current Contents
- ISI's Journal Citation Reports
- Ulrich's International Periodicals Directory
- Books in Print
- Online Public Access Catalogs (OPACs)
- Serial Catalogs
- Publisher Full-Text Databases (Elsevier, Springer-Verlag, IEEE, etc.)

The number of these services available at a library depends on the nature of the library, their budget, customer profile, and other factors. The list of services is dynamic, with services being added and deleted as they become available, fall into disuse or move. Given all this, static linking between an object and the services applicable to an object is not feasible. SFX provides a dynamic lookup of the services that are likely to be available, given the nature of the bibliographic information and a set of heuristics defined by the local library. For example, a book should produce links to "Books in Print" and perhaps "Amazon.com", but not "Journal Citation Reports".

The SFX reference linking service was placed in buckets by way of using "SFX buttons". A button was available for both pre- and post-publication versions of the work, if both versions were known to be available. Figure 23 shows a UPS bucket with

both pre- and post-publication SFX buttons. Of the six constituent archives comprising UPS, only arXiv, RePEc and NCSTRL received SFX buttons. The SFX server did not have enough interesting services to warrant SFX buttons for the buckets from the other three archives. The buttons themselves link to a SFX server, which then queries the calling bucket to retrieve the bucket's metadata in ReDIF format. The SFX server then presents an interface to the user showing the various services that are applicable to the bucket (Fig. 24). The user can correct misspellings in authors' names, volume numbers or other fields that may have been parsed incorrectly before submitting the request to get that service.

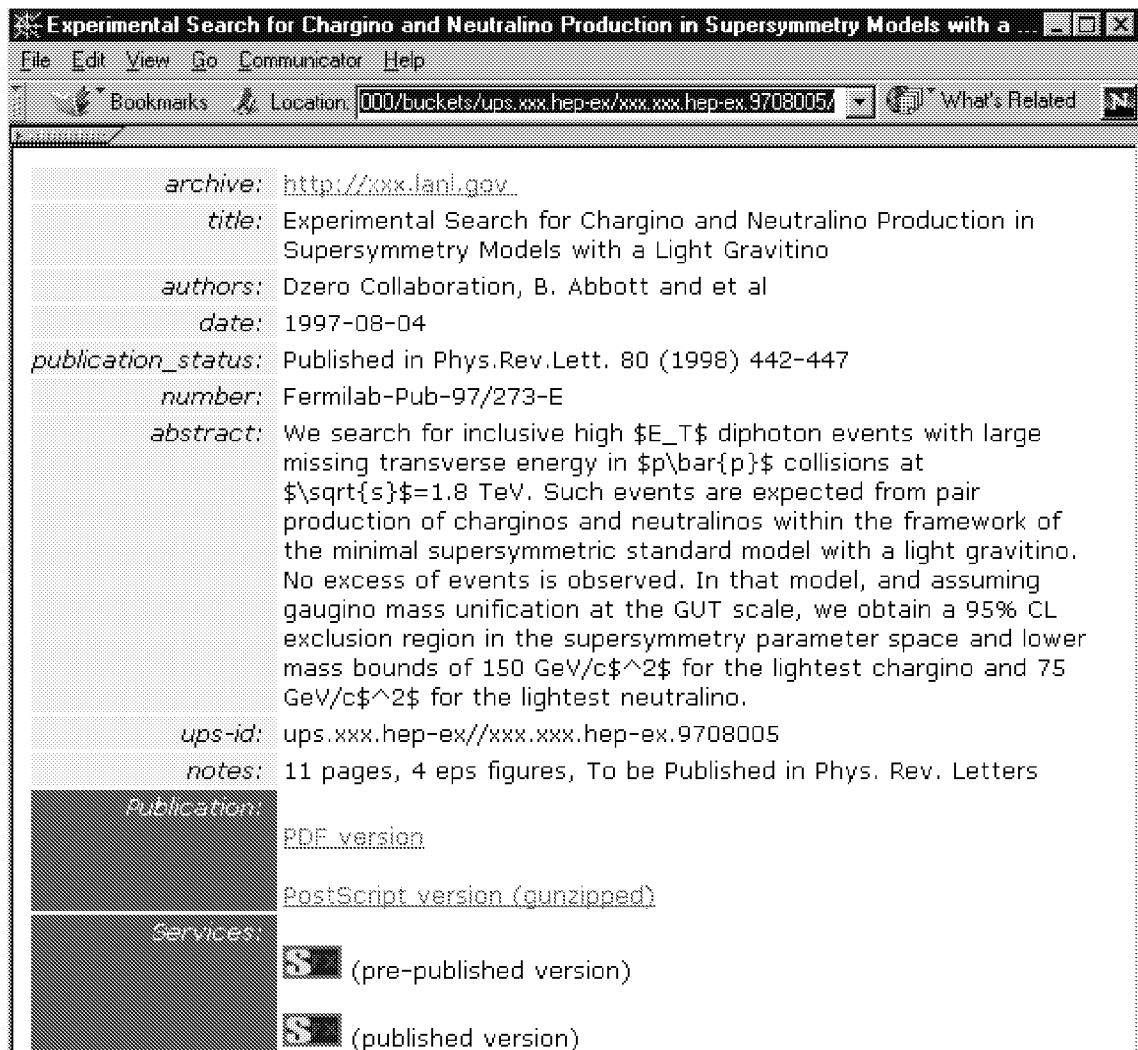


FIG. 23. A UPS bucket with SFX buttons.

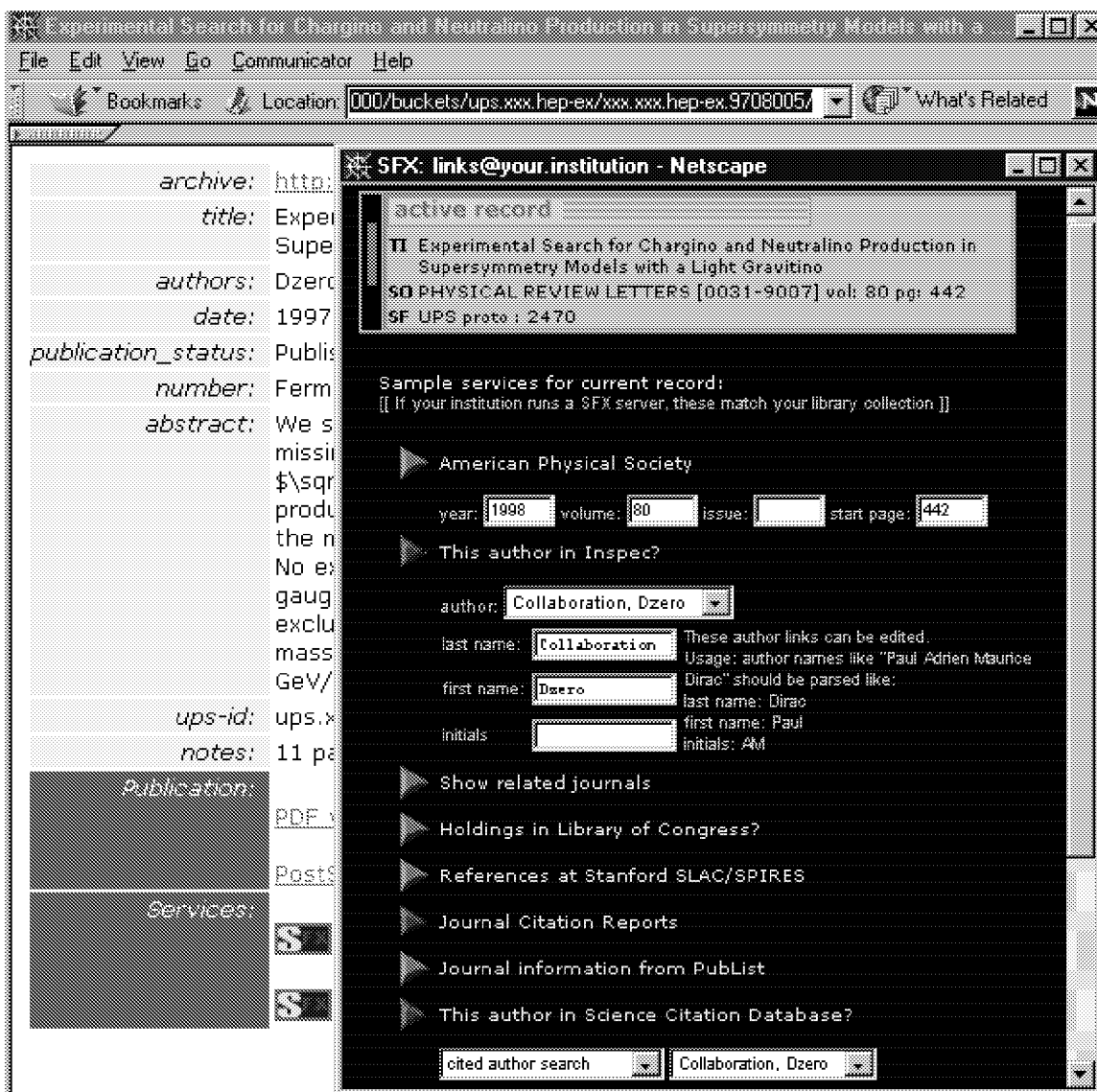


FIG. 24. SFX interface.

Since a SFX server provides an interface to a locally defined set of value-added services (which are often subscription based), each local site is expected to have its own SFX server. This introduces the complication of having to tell the bucket which SFX server a particular user should be referencing. Buckets can set a default SFX server through a bucket preference. SFX server values can also be passed in as arguments to the “display” method, or by using http cookies. The order of precedence is:

1. http argument to the “display” method (“sfx”)
2. http cookie (“sfx_url”)
3. bucket preference (“sfx_server”)

The possible values for the SFX server will be evaluated, and the link to the server is dynamically built in the HTML display to the user. In the UPS prototype, the NCSA http server required by the Dienst software did not support cookies, so a bucket preference was used to specify a SFX server hosted by the University of Ghent for the duration of the demonstration. It would normally be the responsibility of the DL software to set either the cookie, or pass in the argument to the “display” method to correctly specify the SFX server. If this is not possible, the University of Ghent has developed “cookie pusher” scripts that allow a client to overcome the limitation of http cookies only being sent to the site that set them. Using a cookie pusher, a client could use a cookie to point to their local SFX server even when visiting previously unvisited, non-local buckets.

All other SFX demonstrations have involved the modification of the DL software to present SFX buttons during the searching and displaying of results. The UPS implementation of SFX reference linking demonstrates that buckets can be used as mount points for value added services, including those developed by other research groups, and requiring little or no modification of the DL software. This is especially important if the DL software is a commercial, non-open source product. The value-added services are attached to the data object itself, so no matter how the bucket is discovered, the services will be available to the user.

CHAPTER SEVEN

RELATED WORK

7.1 Aggregation

There is extensive research in the area of redefining the concept of “document” or providing container constructs. In this section we examine some of these projects and technologies that are similar to buckets.

7.1.1 Kahn/Wilensky Framework and Derivatives

Buckets are most similar to the digital objects first described in the Kahn/Wilensky Framework (Kahn & Wilensky, 1995), and its derivatives such as the Warwick Framework containers (Lagoze, Lynch, & Daniel, 1996) and its follow-on, the Flexible and Extensible Digital Object Repository Architecture (FEDORA) (Daniel & Lagoze, 1997). In FEDORA, DigitalObjects are containers, which aggregate one or more *DataStreams*. DataStreams are accessed through an Interface, and an Interface may in turn be protected by an Enforcer. Interaction with FEDORA objects occurs through a Common Object Request Broker Architecture (CORBA) (Vinoski, 1997) interface. No publicly accessible, FEDORA implementations is known to exist at this point, and it is not known what repository or digital library protocol limitations will be present.

7.1.2 Multivalent Documents

Multivalent documents (Phelps & Wilensky, 2000) appear similar to buckets at first glance. However, the focus of multivalent documents is more on expressing and managing the relationships of differing “semantic layers” of a document, including language translations, derived metadata, annotations, etc. One of the more compelling demonstrations of Multivalent documents is with geospatial information, with each valence representing features such as rivers, political boundaries, road infrastructure, etc. There is not an explicit focus on the aggregation of several existing data types into a single container. Multivalent documents provide a unique environment for interacting with

information that maps well to the semantics of having multiple “layers”. Although not yet attempted, Multivalent documents could reside inside buckets, effectively combining the benefits of both technologies.

7.1.3 Open Doc and OLE

OpenDoc (Nelson, 1995) and OLE (and its many variations) (Brockschmidt, 1995) are two similar technologies that provide the capability for compound documents. Both technologies can be summarized as viewing the document as a loose confederation of different embedded data types. The focus on embedded documents is less applicable to our digital library requirements than that of a generic container mechanism with separate facilities for document storage and intelligence. OpenDoc and OLE documents are more suitable to be elements within a bucket, rather than a possible bucket implementation.

7.1.4 Metaphoria

Metaphoria is a WWW object-oriented application in which content is separated from the display of content (Shklar, Makower, Maloney, & Gurevich, 1998). Metaphoria is implemented as Java servlets that aggregate derived data sources from simple data sources, with possible multiple layers of derived data sources. A simple data source could be an ASCII file, a WWW page, or an SQL query. Metaphoria parses the content and makes it available through multiple representations, or document object models. It has additional presentation enriching capabilities, such as caching and session management. Metaphoria provides a complex server environment where the main focus is the dynamic reconstitution and presentation of data sources. As such, Metaphoria could sit “above” the bucket layer, where it would be used as a highly sophisticated presentation mechanism for viewing collections of buckets.

7.1.5 VERS Encapsulated Objects

The Victorian Electronic Record Strategy (VERS) focuses on VERS Encapsulated Objects (VEOs) as a way of preserving the governmental records of Australian state of Victoria (Waugh, Wilkinson, Hills, & Dellóro, 2000). VEOs are designed to insure the long-term survivability of the archived object, with as much encapsulation and textual

encoding of its contents as possible, even going as far as expressing binary data formats in Base64 encoding (Borenstein & Freed, 1993). A significant difference between buckets and VEOs is the latter are purely for archival preservation. VEOs are actually XML objects, and thus have no computational capability of their own. They rely on another service to instantiate and read them.

7.1.6 Aurora

The Aurora architecture defines a framework for using container technology to encapsulate content, metadata and usage (Marazakis, Papadakis, & Papadakis, 1998). Aurora defines the containers in which arbitrary components can execute, providing a variety of potential services ranging from shared workspaces, pipelining of electronic commerce components, and workflow management. Aurora's encapsulation of metadata, data and access is similar to that of buckets. The Aurora framework of services are defined in terms of a CORBA-based implementation, and the range of services available in Aurora reflect the richness and complexity of CORBA.

7.1.7 Electronic Commerce

Two representative electronic commerce (or e-commerce) solutions are "DigiBox" (Sibert, Bernstein, & Van Wie, 1995) and IBM's "cryptolopes" (Kohl, Lotspiech, & Kaplan, 1997). Cryptolopes define a three-tier architecture designed to provide potential anonymity between both the users and providers of information through use of a middle layer clearinghouse. The goal of DigiBox is "to permit proprietors of digital information to have the same type and degree of control present in the paper world" (Sibert, Bernstein, & Van Wie, 1995). As such, the focus of the DigiBox capabilities are heavily oriented toward cryptographic integrity of the contents, and not on the less stringent demands of the current average digital library.

E-commerce solutions are highly focused on providing "superdistribution" (Mori & Kawahara, 1990), where information objects are opaque and can be distributed widely, but are only fully accessible through use of a key (presumably for sale from a service).

There appear to be no hooks for DigiBox or cryptolope intelligence. Both are commercial endeavors and are less suitable for research in value-added DL services.

7.1.8 Filesystems and File Formats

To a lesser extent, buckets are not unlike some of the proposals from various experimental filesystems and scientific data types. The Extensible File System (ELFS) (Karpovich, Grimshaw, & French, 1994) provides an abstract notion of “file” that includes both aggregation, data format heterogeneity, and high performance capabilities (striping, pre-fetching, etc.). While ELFS is designed primarily for a non-DL application (i.e., high-performance computing), it is typical of an object-oriented approach to file systems, with generic access APIs hiding the implementation details from the programmer.

The Hierarchical Data Format (HDF) and similar formats (netCDF, HDF-EOS, etc.) is a multi-object, aggregative data format that is alternatively: raw file storage, the low-level I/O routines to access the raw files, an API for higher level tools to access, and a suite of tools to manipulate and analyze the files (Stern, 1995). While HDF is mature and has an established user base, it is largely created by and for the earth and atmospheric sciences community, and this community’s constraints limits the usefulness of HDF as a generalized DL application. It is worth noting, however, that buckets of HDF files should be entirely possible and appropriate.

7.2 Intelligence

Intelligent agent research is an active area. There are many different definitions of what constitutes an “agent”. From Birmingham (1995), we use the following definition:

“Autonomy: the agent represents both the capabilities (ability to compute something) and the preferences over how that capability is used. [...]

Negotiation: since the agents are autonomous, they must negotiate with other agents to gain access to other resources or capabilities. [...]

Using this definition, it is clear that buckets satisfy the autonomy condition, since buckets perform many computational tasks that are influenced by their individual preferences. However, the current implementation of buckets only weakly satisfy the negotiation condition, since only a handful of transactions have actual negotiation. An example of such a transaction is the case when a bucket requests metadata conversion from the BCS; there is a negotiation phase where the requesting bucket and the BCS server negotiate the availability of metadata formats. However, the direction is clear that buckets are becoming increasingly intelligent, so they will eventually be considered unequivocally as true agents.

In practice, the information environment application of intelligent agents has generally dealt with assistants to aid in searching, search ordering, finding pricing bargains from on-line sales services, calendar maintenance, and other similar tasks. Birmingham (1995) defines the three classes of agents in the University of Michigan section of the NSF funded DLI: User Interface Agents, Mediator Agents, and Collection Interface Agents. Other projects are similar: agents to help DL patrons (Sanchez, Legget, & Schnase, 1997), retrieval and access agents (Salampasis, Tait, & Hardy, 1996), and DL construction/authoring (Sanchez, Lopez, & Schnase, 1998). There appear to be no other projects that attempt to make archival objects intelligent. Note that making the archived object intelligent does not preclude the use of other agents in a DL environment (search agents, collection agents, etc.). In fact, increasingly intelligent buckets should be able to assist the traditional DL agents in performing their tasks.

7.3 Archives

There has been an increased amount of interest regarding the nature of archives, specifically in the separation of the roles of providing (or “publishing”, or “archiving”) data and of discovering (or “searching”) data. In early DL projects, there was often little distinction but with DLs reaching larger scales and the greater interest in interoperability, such vertically integrated DLs are no longer feasible. The current highest profile archive

project is the Open Archives initiative, presented in Chapter Six. However, a number of other projects have demonstrated this separation of roles as well.

The Guildford Protocol (Cruz & Krichel, 1999) has been in use in the Economics community for quite some time within the RePEc project (a participant in UPS). RePEc is unique in that it specifies no user services, but only focuses on the coordination and propagation of distributed collections of metadata. Several DLs have been built from the metadata harvested from RePEc. Dienst has a Repository service, a portion of which forms the basis for the OAI harvesting protocol. Unfortunately, the full Dienst Repository service also contains the concept of a document model. The inclusion of a document model in an archival service is too heavy and limits the applications of that archival service. Stanford has proposed archival awareness algorithms, in which distributed archives can maintain consistency in the face of updates and deletions (Crespo & Garcia-Molina, 1997).

There are a number of other possible implementations for archival services. Although using a RDBMS or the light-weight directory access protocol (LDAP) (Yeong, Howes, & Kille, 1995) seems to be an obvious implementation, there appear to be no such archive implementation within the DL community.

7.4 Bucket Tools

Related to buckets, but being developed separately by another team at Old Dominion University, are tools for bucket creation, administration and simple workflow management. The need for high quality tools for the creation and management of buckets is obvious: no matter how expedient and useful buckets may be, if they are not created in sufficient quantities they will not be adopted on a large scale. The bucket tools are needed to hide the details of bucket creation from ordinary users, and if bucket tools use the bucket API for all of their actions, the tools should be applicable across bucket implementations.

For batch creation of buckets, a number of scripts have been developed to automate the process. However, they tend to be specialized for the DL they are




populating and are not really worthy of being called a “tool”. A suite of three integrated tools has been created that can be installed at a local site to provide a simple author-based publishing workflow. These tools should be considered a reference implementation of possible tools for buckets – additional implementations or tools of different natures are possible. The process begins with an author accessing the Creation Tool (Fig. 25), which is used to create and populate a bucket. The buckets are kept in a temporary staging archive, where only the author can access the bucket. The population can occur over many sessions, with the author saving the intermediate bucket at the end of each session.

Netscape: Digital Library

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Bookmarks Location: <http://res-ups.cs.odu.edu:8888/tool/creation/le/> What's Related

 **NCSTRL+ Creation Tool**  
NASA and Old Dominion University

Edit Add Delete Comments Save Submit Help Home

Flight Experiments
data files
documentation
NewElement

Edit Element Metadata

Title: preliminary report

Authors(Separated by semicolon): B. Johnson; H. Smith

Authors' Affiliation:

Subject:

Subject Classes:

Subject Divisions:

Archival Type:

FIG. 25. Creation tool.

When the author is done with the bucket, it can be submitted for review by management. This physically moves the bucket from its temporary staging archive to the management archive. The manager receives an email stating that a new bucket has arrived. The manager will then use the Management Tool (Fig. 26) to review the pending buckets. The manager can approve, disapprove or defer action on the list of buckets. Disapproved buckets will go back to the author's temporary staging archive, and approved buckets will be promoted to the site's official archive. It is the official archive that DLs know to harvest from, and placement into this privileged archive constitutes "publishing".

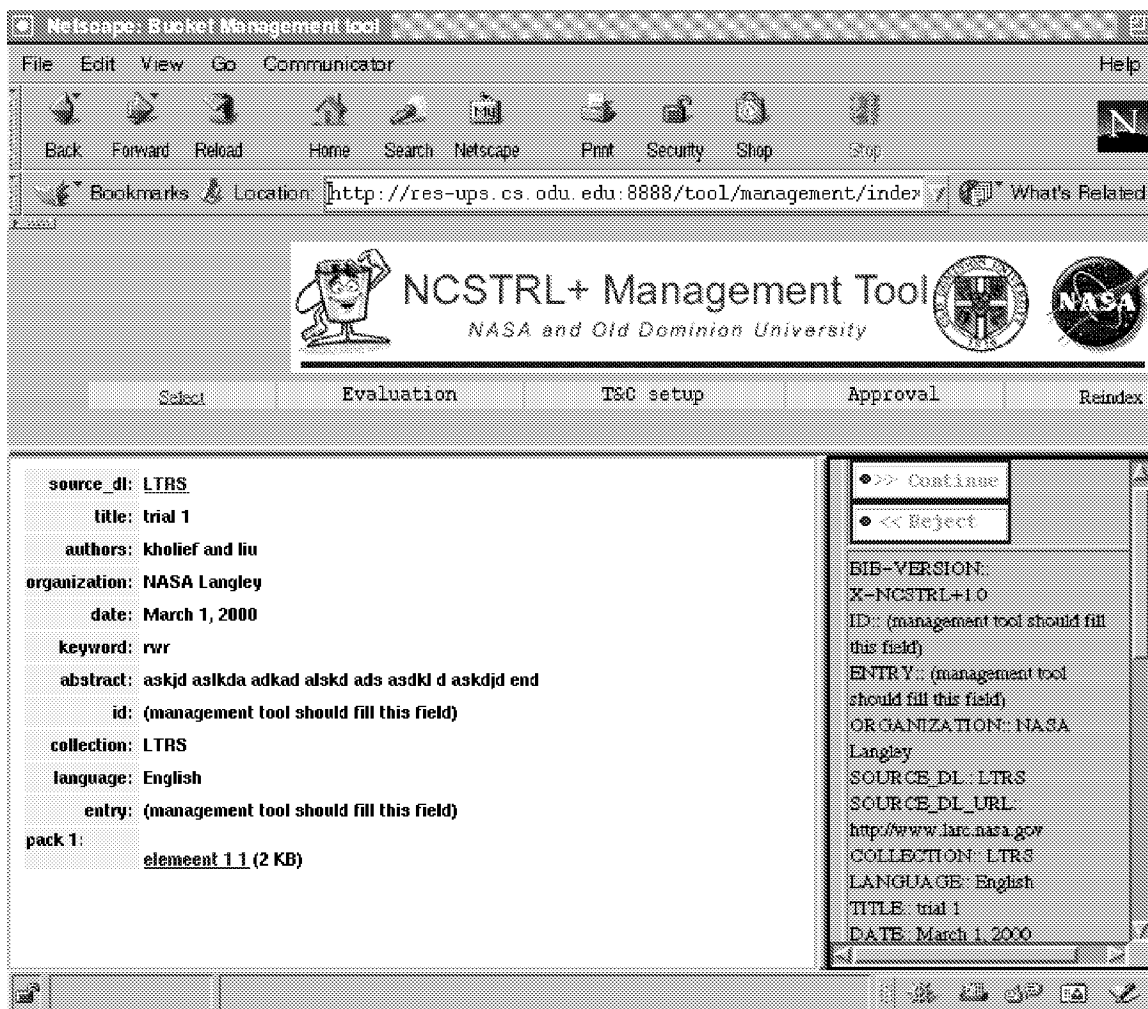


FIG. 26. Management tool.

Finally, the Administration Tool (Fig. 27) provides a mechanism for the maintenance of already published buckets. This tool provides a standard interface to multi-cast messages to multiple buckets, for such functions as harvesting their log files, updating preferences, adding general value-added services, and all such functions that are not part of the creation/approval process.

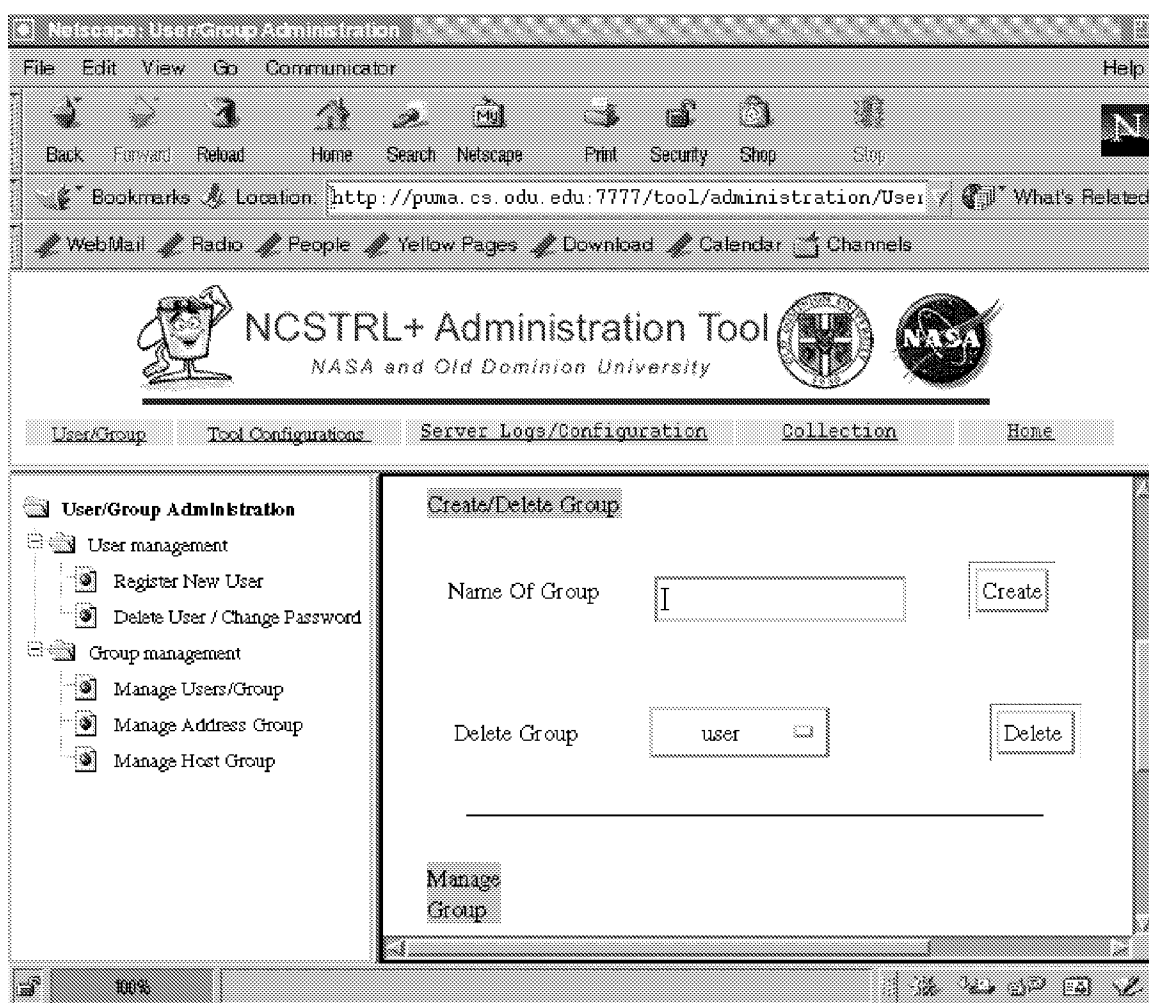


FIG. 27. Administration tool.

CHAPTER EIGHT

FUTURE WORK

8.1 Alternate Implementations

The lessons learned from implementing buckets and the supporting technology, along with their success and popularity in NCSTRL+ and UPS, point to many areas of future work, of both practical and academic interest. Perl and CGI are good development platforms. Perl is expressive, commonly available and reasonably fast. However, there are a number of other interesting languages that deserve study. Plain CGI is low performance, often running at 10% - 50% of the performance of regular http requests for small transactions, with Perl CGI programs performing slightly worse than C CGI programs (McGrath, 1996).

8.1.1 Buckets

There are three, possibly overlapping, areas for alternate implementation for buckets: rewriting buckets in another language, replacing CGI, and making the API available through something other than http.

Buckets could be rewritten in another scripting language like Python (Lutz, 1996) or Tcl (Ousterhout, 1994). Buckets currently exploit some of the Perl idioms, and it is possible that Python, Tcl or another similar language could easily add new capabilities. An obvious area to explore is writing a Java (Arnold & Gosling, 1996) implementation of buckets – given the proliferation of Java compilers embedded in a variety of applications and even hardware devices. The run-time performance gains of implementing buckets in a compiled language such as C/C++ would probably be negated by the lack of mobility resulting from “compiled buckets”.

Replacing CGI would be a big performance win in high-traffic DLs. One example relevant to Perl buckets is that they could be written to use `mod_perl`, where the Perl interpreter is actually embedded into the Apache http server (Stein, MacEarchern, Mui,

1999). This eliminates the costly overhead of CGI. Similarly, Java buckets could use servlets (Hunter, Crawford, & Ferguson, 1998) for performance gains.

If the bucket API was available to running programs through a mechanism more familiar and natural for applications programs, such as CORBA, then the capabilities of buckets would be greatly expanded. It would be easier for running programs to automatically retrieve and fill buckets as they are running without converting their requests to http. Probably the simplest way to provide this capability would be through a CORBA gateway that converted messages to/from bucket methods.

8.1.2 Dumb Archives

The current implementation of an archive service is a modified bucket which has its methods extended to include basic set management functionality. While it would be possible to add more sophisticated methods to DA, and the orthogonal approach of using a modified bucket has an aesthetic appeal, it is also possible to implement archive services with other technologies as well. DA functionality could be implemented using a RDBMS or LDAP. The OAi harvesting protocol (Van de Sompel & Lagoze, 2000) is a subset of the Dienst protocol, including only those verbs related to archive management. The proposed collection service for Dienst (Lagoze & Fielding, 1998) could also be the foundation for an implementation of the Smart Object, Smart Archive (SOSA) model.

8.1.3 Bucket Communication Space

While alternate archive implementations could be constructed or adapted without requiring specific knowledge about buckets, this would be more difficult for the BCS, since it performs services specific to buckets. A modified bucket might not be the most efficient implementation for the BCS, but unlike for the DA, there are no obvious candidates for alternate BCS implementations.

8.2 Extended Functionality

Contrasting with implementing the same functionality in different languages or environments, there are a number of new functionalities that could be implemented in the

short term. These include using pre-defined bucket packages and elements and XML metadata support.

8.2.1 Pre-defined Packages and Elements

Some functionality improvements could be made not through new or modified methods, but through conventions established on the current infrastructure. One convention already adopted was the use of a `BCS_Similarity.pkg` package to hold the resulting links of the BCS similarity indexing. Other possible uses include: standard element names for bucket checksums (entire bucket, packages or elements) to insure the integrity of elements; standard packages (or elements) for bibliographic citation information, possibly in multiple encodings; or standard package or element names for previous revisions of bucket material. Conventions are likely to be adopted as need and applications arise.

8.2.2 XML Metadata

Perhaps the most urgent improvement for buckets involves removing the RFC-1807 historical dependency and using XML as its canonical metadata set. The current modified RFC-1807 format dictates a generic, but inflexible two-level bucket structure. A challenge for XML support is that until XML parsers are ubiquitously available (e.g., included in the Perl standard library), the bucket will have to internally be able to parse XML files. Switching to XML should allow the modeling of arbitrarily complex data objects, and the use of a “bucket style sheet” should allow the display method to more easily change the bucket presentation without specific code changes in the “display” method itself. XML support would also allow the use of the Resource Description Framework (RDF) (Miller, 1998). RDF would allow for greater metadata expressiveness, facilitating the sharing of semantic metadata models within a standard framework.

Adding lightweight XML parser support, along with style sheets and generalized handling of metadata will be a difficult task. All of this is possible, but the parser must be compact enough to not greatly increase the bucket’s storage requirement and not introduce dependencies that would damage the mobility and self-sufficiency of buckets.

8.2.3 More Intelligence

There are a number of functions that buckets for which buckets already have hooks in place, but have not yet been fully automated. For example, the “lint” method can detect internal errors and misconfigurations in the bucket, but it does not yet attempt to repair a damaged bucket. Similarly, a bucket preference could control the automatic updating of buckets when new releases are available, while still maintaining the bucket’s own configuration and local modifications. The updated bucket could then be tested for correct functionality, and rolled back to a previous version if testing fails. The option of removing people from the bucket update cycle would ease a traditional administration burden.

Buckets could also be actively involved in their own replication and migration, as opposed to waiting for human intervention for direction. Buckets could copy themselves to new physical locations so they could survive physical media failures, existing either as functioning or dormant replicates. Should the canonical bucket be “lost” somehow, buckets could vote among themselves to establish a new priority hierarchy. Distributed storage projects such as the Archival Intermemory (Goldberg & Yianilos, 1998) or Internet2 Distributed Storage Infrastructure Project (Beck & Moore, 1998) could serve as complementary technologies for implementing migratory buckets.

8.3 Security, Authentication and Terms & Conditions

While every effort has been made to make buckets as secure and safe as possible, a full-scale investigation by an independent party has not been performed. A first level of investigation would be in attacking the buckets themselves, to determine if the buckets could be damaged, made to perform actions prohibited by their T&C files, or otherwise be compromised. A second level of investigation would be examining if buckets could be compromised through side effects resulting from attacks on other services. Currently, buckets have no line of defense if the http server or the system software itself is attacked. Having buckets employ some sort of encryption on their files that is decoded dynamically would offer a second level of security, making the buckets truly opaque data

objects that could withstand at least some level of attack if the system software was compromised.

Authentication is currently done through the standard http procedures, relying on the server to correctly set the value of `REMOTE_ADDR`, `REMOTE_HOST`, and `REMOTE_USER`. Authentication alternatives using Kerberos (Steiner, Neuman, & Schiller, 1988), MD5 (Rivest, 1992), or X.509 (CCIT, 1998) should be explored so buckets can fit into a variety of large-scale authentication schemes in use at various facilities.

The T&C model used now is simple, and does not allow for complex expressions, especially nesting of directives. For example, a `display.tc` file could contain:

```
user:nelson
user:maly
host:*.larc.nasa.gov
host:*.cs.odu.edu
package:datasets.pkg
package:software.pkg
```

The above imposes all the user and host restrictions on both packages. There is currently no way to have different restrictions on different packages or elements within a single T&C file. XML is a natural format to implement the upgraded T&C files, so that a T&C file could be constructed to easily describe the hierarchy and relationship of the desired T&C. An example would look something like:

```
<?xml version="1.0"?>
<package name="datasets.pkg">
  <user name="nelson"\>
  <host name="*.larc.nasa.gov"\>
</package>
<package name="software.pkg">
  <user name="maly"\>
  <host name="*.cs.odu.edu"\>
</package>
```

8.4 New Applications

Buckets provide a base level of functionality that is immediately useful. However, reflection on the capabilities buckets provide soon causes one to think of the additional capabilities that buckets could provide in the future.

8.4.1 Discipline-Specific Buckets

Buckets are currently not specific to any discipline; they have a generic “one-size-fits-all” approach. While this is attractive for the first generation of buckets since it excludes no disciplines, it also does nothing to exploit assumptions and extended features of a specific discipline. Intuitively, an earth science bucket could have different requirements and features than a computational science bucket. Given a scientific discipline, it could be possible to define special data structures and even special methods or method arguments for the data, such as geo-spatial arguments retrieving data from earth-science buckets or compilation services for a computational science bucket.

Generalized XML support in the bucket (discussed above) would simplify tailoring buckets to different ontologies. Buckets could begin as generic buckets, then acquire specific “skins” (in computer-game parlance) that would dictate their look and feel as well as their functionality.

8.4.2 Usage Analysis

There are several DL projects that focus on determining the usage patterns of their holdings and dynamically arranging the relationships within the DL holdings based on these patterns (Bollen & Heylighen, 1997; Rocha, 1999). All of these projects are similar in that they extract usage patterns of passive documents, either examining the log files of the DL, or instrumenting the interface to the DL to monitor user activity, or some hybrid of these approaches. An approach that has not been tried is for the objects themselves to participate in determining the usage patterns, perhaps working in conjunction with monitors and log files. Since the buckets are executable code, it is possible to not just instrument the resource discovery mechanisms, but the archived objects also. We have

experience instrumenting buckets to extract additional usage characteristics, but we have not combined this strategy with that of the other projects.

8.4.3 Software Reuse

Buckets could have an impact in the area of software reuse as well. If a bucket stores code, such as a solver routine, it would not have to be limited to a model where users extract the code and link it into their application, but rather the bucket could provide the service, and be accessible through remote procedure call (RPC)-like semantics. Interfaces between distributed computing managers such as Netsolve (Casanova & Dongarra, 1998) or NEOS (Czyzyk, Mesnier, & More) and “solver buckets” could be built, providing simple access to the solver buckets from running programs. Data, and the routines to derive and manipulate it, could reside in the same bucket in a DL. This would likely be tied with a discipline specific application, such as a bucket having a large satellite image and a method for dynamically partitioning and disseminating portions of the data.

Or users could temporarily upload data sets into the bucket to take advantage of a specialized solver resident within the bucket without having to link it into their own program. This would be especially helpful if the solver had different system requirements, and it could not easily be hosted on a user’s own machine. However, the traditional model of “data resides in the library; analysis and manipulation occurs outside the library” can be circumvented by making the archived objects also be computational objects.

CHAPTER NINE

RESULTS AND CONCLUSIONS

Buckets were born of our experience in creating, populating and maintaining several production DLs for NASA. The users of NASA DLs repeatedly wanted access to data types beyond that of the technical publication, and the traditional publication systems and the digital systems that automated them were unable to adequately address their needs. Instead of creating a raft of competing, “separate-but-equal” DLs to contain the various information types, a container object was created capable of capturing and preserving the relationship between any number of arbitrary data types.

Buckets are aggregative, intelligent, WWW-accessible digital objects that are optimized for publishing in DLs. Buckets implement the philosophy that information itself is more important than the DL systems used to store and access information. Buckets are designed to imbue the information objects with certain responsibilities, such as the display, dissemination, protection and maintenance of its contents. As such, buckets should be able to work with many DL systems simultaneously, and minimize or eliminate the necessary modification of DL systems to work with buckets. Ideally, buckets should work with everything and break nothing. This philosophy is formalized in the SODA (Smart Object, Dumb Archive) DL model. The objects become “smarter” at the expense of the archives (who become “dumber”), as functionalities generally associated with archives are moved into the data objects themselves. This shift in responsibilities from the archive into the buckets results in a greater storage and administration overhead, but these overheads are small in comparison to the great flexibility that buckets bring to DLs.

This research has successfully met the objectives as stated in Chapter Two. First, the concept of “buckets” was introduced, which is the collection of mechanisms and protocols for aggregating and mobilizing content and services on the content. A well-

defined bucket API (Appendix B) is the result of previous DL experience and study of the bucket concept. Secondly, a reference implementation of buckets was developed that is written in Perl that uses http and CGI mechanisms for transport of bucket messages. This reference implementation fully implements the bucket API. Other research projects are investigating implementing the bucket API using other technologies, including Java servlets and the Oracle RDBMS. Lastly, the bucket concept and the Perl-based reference implementation were demonstrated in a variety of application and DL deployments. Research project buckets, university class buckets, and traditional STI publication buckets were created. STI publication buckets were demonstrated in great numbers in the NCSTRL+ and UPS experimental DLs. To facilitate the adoption of buckets, other projects have introduced support tools for buckets, most notably a Creation Tool, Management Tool and Administration Tool.

Buckets have demonstrated their flexibility in a number of ways. First, for the UPS project “light-weight buckets” emerged as a useful variation of buckets and it was easy to augment buckets with value-added services such as the SFX reference linking service, and then later similarity matching links. The extensibility of buckets was further demonstrated when the creation of archive services (DA) and the Bucket Communication Space were implemented using modified buckets. Since any ordinary bucket could be turned into a DA or BCS bucket through a well-defined series of transformations, this approach showed the orthogonality of buckets in a variety of applications. Buckets are general purpose, stand-alone, WWW-accessible DL workhorses.

There are a number of projects that have similar aggregation goals as buckets. Some are from the DL community, and others are from e-commerce and computational science. Most do not have the SODA-inspired motivation of freeing the information object from the control of a single server. The mobility and independence of buckets are not seen in other DL projects. Most DL projects that focus on intelligence or agency are focused on aids to the DL user or creator; the intelligence is machine-to-human based.

Buckets are unique because the information objects themselves are intelligent, providing machine-to-machine (or, bucket-to-bucket) intelligence.

Buckets are already having a significant impact in how NASA and other organizations such as Los Alamos National Laboratory, Air Force Research Laboratory, Old Dominion University, and the NCSTRL steering committee are designing their next generation DLs. The interest in buckets has been high, and every feature introduced seems to raise several additional areas of investigation for new features and applications. First and most important, the creation of high quality tools for bucket creation, management and maintenance in a variety of application scenarios is absolutely necessary. Without tools, buckets will not be widely adopted. Other short-term areas of investigation include optimized buckets, alternate implementations of buckets, discipline-specific buckets, XML support, and extending authentication support to include a wider variety of technologies. Long-range plans include significant utilization of bucket mobility and bucket intelligence, including additional features in the Bucket Communication Space. Buckets, through aggregation, intelligence, mobility, self-sufficiency, and heterogeneity, provide the infrastructure for information object independence. The truly significant applications of this new breed of information objects remain undiscovered.

REFERENCES

- Adler, S., Berger, U., Bruggemann-Klein, A., Haber, C. Lamersdorf, W., Munke, M., Rucker, S. & Spahn, H. (1998). Grey literature and multiple collections in NCSTRL. In A. Barth, M. Breu, A. Endres & A. de Kemp (eds.), *Digital libraries in computer science: the MeDoc approach* (pp. 145-170), Berlin: Springer.
- Andreoni, A., Bruna Baldacci, M., Biagioni, S., Carlesi, C., Castelli, D., Pagano, P. & Peters, C. (1998). Developing a European technical reference digital library. In S. Abiteboul & A.-M. Vercoustre (eds.), *Research and advanced technology for digital libraries, third European conference, ECDL '99* (pp. 343-362), Berlin: Springer.
- Arms, W. A. (1999). Preservation of scientific serials: three current examples. *Journal of Electronic Publishing*, 5(2). Available at <http://www.pres.umich.edu/jep/05-02/arms.html>
- Arnold, K. J., & Gosling, J. (1996). *The Java programming language*. Reading, MA: Addison-Wesley.
- Baker, B. S. (1995a). On finding duplication and near-duplication in large software systems. *Proceedings of the second IEEE working conference on reverse engineering* (pp. 86-96), Toronto, Canada. Available at <http://cm.bell-labs.com/cm/cs/doc/95/2-bsb-3.pdf>
- Baker, M. (1995b). Cluster computing review. Syracuse University Technical Report NPAC SCCS-748. Available at <http://www.npac.syr.edu/techreports/html/0700/abs-0748.html>
- Beck, M. & Moore, T. (1998). The Internet2 distributed storage infrastructure project: an architecture for Internet content channels. *Computer Networking and ISDN Systems*, 30(22-23), 2141-2148. Available at <http://dsi.internet2.edu/pdf-docs/i2-chan-pub.pdf>
- Bennington, J. (1952). The integration of report literature and journals. *American Documentation*, 3(3), 149-152.
- Bennion, B. C. (1994, February/March). Why the science journal crisis? *ASIS Bulletin*, 25-26.

- Berners-Lee, T., Cailliau, R., Groff, J.-F., & Pollermann B. (1992). World-Wide Web: the information universe. *Electronic Networking: Research, Applications and Policy*, 2(1), 52-58.
- Birmingham, W. P. (1995). An agent-based architecture for digital libraries. *D-Lib Magazine*, 1(7) July 1995. <http://www.dlib.org/dlib/July95/07birmingham.html>
- Bollen, J. & Heylighen F. (1997). Dynamic and adaptive structuring of the World Wide Web based on user navigation patterns. *Proceedings of the Flexible Hypertext Workshop* (pp. 13-17), Southampton, UK. Available at <http://www.c3.lanl.gov/~jbollen/pubs/Bollen97.htm>
- Bookstein, A. & Swanson, D. R. (1974). Probabilistic models for automatic indexing. *Journal of the American Society for Information Science*, 25, 312-319.
- Borenstein, N. & Freed, N. (1993). MIME (multipurpose Internet mail extensions) part one: mechanisms for specifying and describing the format of Internet message bodies. Internet RFC-1521. Available at <ftp://ftp.isi.edu/in-notes/rfc1521.txt>
- Brenner, S. (2000). The cgi-lib.pl homepage. Available at <http://cgi-lib.berkeley.edu/>
- Brockschmidt, K. (1995). *Inside OLE 2*. Redmond, WA: Microsoft Press.
- Carriero, N. & Gelernter, D. (1989). Linda in context. *Communications of the ACM*, 32(4), 444-458.
- Casanova, H. & Dongarra, J. (1998). Applying Netsolve's network-enabled solver. *IEEE Computational Science & Engineering*, 5(3), pp. 57-67.
- CCITT (1998). The directory authentication framework. CCITT Recommendation X.509.
- Crespo, A. & Garcia-Molina, H. (1997). Awareness services for digital libraries. In C. Peters & C. Thanos (eds.), *Research and advanced technology for digital libraries, first European conference, ECDL '97* (pp. 147-171), Berlin: Springer.
- Croft, W. B. & Harper, D. J. (1979). Using probabilistic models of document retrieval without relevance information. *Documentation*, 35(4), 285-295.
- Cruz, J. M. B. & Krichel, T. (1999). Cataloging economics preprints: an introduction to the RePEc project. *Journal of Internet Cataloging*, 3(2-3).

- Czyzyk, J., Mesnier, M. P. & More, J. J. (1998). The NEOS solver. *IEEE Computational Science & Engineering*, 5(3), pp. 68-75.
- Daniel, R. & Lagoze, C. (1997). Distributed active relationships in the Warwick framework. *Proceedings of the second IEEE metadata workshop*, Silver Spring, MD.
- Davis, J. R. & Lagoze, C. (1994). A protocol and server for a distributed digital technical report library. *Cornell University Computer Science Technical Report TR94-1418*. Available at <http://ncstrl.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR94-1418>
- Davis, J. R., Fielding, D., Lagoze, C. & Marisa, R. (2000). The Santa Fe convention: the Open Archives Dienst subset. Available at http://www.openarchives.org/sfc/sfc_dienst.htm
- Davis, J. R. & Lagoze, C. (2000). NCSTRL: design and deployment of a globally distributed digital library. *Journal of the American Society for Information Science*, 51(3), 273-280.
- Esler, S. L. & Nelson, M. L. (1998). Evolution of scientific and technical information distribution. *Journal of the American Society for Information Science*, 49(1), 82-91. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/1998/jp/NASA-98-jasis-slc.pdf>
- Fielding, R., Gettys, J., Mogul J. C., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999). Hypertext transfer protocol – HTTP/1.1. *Internet RFC-2616*. Available at <ftp://ftp.isi.edu/in-notes/rfc2616.txt>
- Finin, T., Fritzson, R., McKay, D. & McEntire, R. (1994). KQML as an agent communication language. *Proceedings of the third international conference on information and knowledge management* (pp. 447-455), Gaithersburg, MD. Available at <http://www.cs.umbc.edu/kqml/papers/kqml-acl.ps>
- Fox, E. A., Eaton, J. L., McMillan, G., Kipp, N. A., Mather, P., McGonigle, T., Schweiker, W. & DeVane, Brian. Networked digital library of theses and dissertations. *D-Lib Magazine*, 3(9). Available at <http://www.dlib.org/dlib/september97/theses/09fox.html>
- Frakes, W. B. & Baeza-Yates, R. (1992). *Information retrieval: data structures & algorithms*. Upper Saddle River, NJ: Prentice-Hall.

- French, J. C., Powell, A. L., Schulman, E. & Pfaltz, J. L. (1997). Automating the construction of authority files in digital libraries: a case study. In C. Peters & C. Thanos (eds.), *Research and advanced technology for digital libraries*, first European conference, ECDL '97 (pp. 55-71), Berlin: Springer.
- Ginsparg, P. (1994). First steps towards electronic research communication. *Computers in Physics*, 8, 390-396.
- Goldberg, A. V. & Yianilos, P. N. (1998). Towards an archival intermemory. *Proceedings of the IEEE forum on research and technology advances in digital libraries* (pp. 147-156), Santa Barbara, CA.
- Gray, D. E. (1953). Organizing and servicing unpublished reports. *American Documentation* 4(3), 103-115.
- Griffin, S. M. (1999). Digital Library Initiative – phase 2. *D-Lib Magazine*, 5/(7-8). Available at <http://www.dlib.org/dlib/july99/07griffin.html>
- Griffiths, J.-M. & King, D. W. (1993). *Special libraries: increasing the information edge*. Washington, DC: Special Libraries Association.
- Halpern, J. Y. & Lagoze, C. (1999). The Computing Research Repository: promoting the rapid dissemination and archiving of computer science research. *Proceedings of the fourth ACM conference on digital libraries* (pp. 3-11), Berkeley, CA.
- Harman, D. (1992). Ranking algorithms. In W. B. Frakes & R. Baeza-Yates (Eds.), *Information retrieval: data Structures & algorithms* (pp. 363-392), Upper Saddle River, NJ: Prentice-Hall.
- Harnad, S. (1997). How to fast-forward serials to the inevitable and the optimal for scholars and scientists. *Serials Librarian*, 30, 73-81. Available at <http://www.cogsci.soton.ac.uk/~harnad/Papers/Harnad/harnad97.learned.serials.html>
- Henderson, A. (1999). Information science and information policy: the use of constant dollars and other indicators to manage research investments. *Journal of the American Society for Information Science*, 50(4), 366-379.
- Hunter, J., Crawford, W. & Ferguson, P. (1998). *Java servlet programming*. Sebastopol CA: O'Reilly & Associates.
- Image Alchemy (2000). Available at <http://www.handmadesw.com/his/specs.html>

- ImageMagick (2000). Available at
<http://www.wizards.dupont.com/cristy/ImageMagick.html>
- Jacobsen, D. (1996). bp, a Perl bibliography package. Available at
<http://www.ecst.csuchico.edu/~jcabosd/bib/bp/>
- Kahle, B., Morris, H., Davis, F., Tiene, K., Hart, C., & Palmer, R. (1992). Wide area information servers: an executive information system for unstructured files, *Electronic Networking: Research, Applications, and Policy*, 2(1), 59-68.
- Kahle, B. (1997). Preserving the Internet. *Scientific American*, 264(3).
- Kahn, Robert E. (1995). An introduction to the CS-TR project. Available at
<http://www.cnri.reston.va.us/home/describe.html>
- Kahn, R. & Wilensky, R. (1995) A framework for distributed digital object services. *cnri.dlib/tn95-01*. Available at
<http://www.cnri.reston.va.us/home/cstr/arch/k-w.html>
- Kaplan, J. A. & Nelson, M. L. (1994). A comparison of queueing, cluster and distributed computing systems. NASA Technical Memorandum 109025. Available at
<http://techreports.larc.nasa.gov/ltrs/PDF/tm109025.pdf>
- Karpovich, J. F., Grimshaw, A. S. & French, J. C. (1994). Extensible file systems (ELFS): an object-oriented approach to high performance file I/O. *Proceedings of the ninth annual conference on object-oriented programming systems, languages and applications* (pp. 191-204), Portland, OR.
- Kohl, U., Lotspiech, J. & Kaplan, M. A. (1997). Safeguarding digital library contents and users. *D-Lib Magazine*, 3(9). Available at
<http://www.dlib.org/dlib/septemeber97/ibm/lotspiech.html>
- Knuth, D. E. (1986). *The TeXbook*. Reading, MA: Addison-Wesley.
- Lagoze, C. & Ely, D. (1995). Implementation issues in an open architectural framework for digital object services. Cornell University Computer Science Technical Report, TR95-1540. Available at
<http://ncstrl.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR95-1540>
- Lagoze, C., Shaw, E., Davis, J. R., & Krafft, D. B. (1995). Dienst: implementation reference manual. Cornell University Technical Report TR95-1514. Available at
<http://ncstrl.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR95-1514>

- Lagoze, C., Lynch C. A., & Daniel, R. (1996). The Warwick framework: a container architecture for aggregating sets of metadata. Cornell University Computer Science Technical Report TR-96-1593. Available at <http://ncstrl.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR96-1593>
- Lagoze, C. & Fielding, D. (1998). Defining collections in distributed digital libraries. D-Lib Magazine, 4(11). Available at <http://www.dlib.org/dlib/november98/lagoze/11lagoze.html>
- Lagoze, C. & Payette, S. (1998). An infrastructure for open-architecture digital libraries. Cornell University Computer Science Technical Report TR98-1690. Available at <http://ncstrl.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR98-1690>
- Lasher, R. & Cohen, D. (1995). A format for bibliographic records. Internet RFC-1807. Available at <ftp://ftp.isi.edu/in-notes/rfc1807.txt>
- Lawrence, S., Bollacker, K. & Giles, C. L. (1999). Distributed error correction. Proceedings of the fourth ACM conference on digital libraries (p. 232), Berkeley, CA.
- Lawrence, S. & Giles, C. L. (1998). Searching the World Wide Web. Science, 280, 98-100. Available at <http://www.neci.nj.nec.com/~lawrence/science98.html>
- Lesk, M. E. (1978). Some applications of inverted indexes on the UNIX system. Bell Laboratories Computing Science Technical Report 69.
- Lesk, M. E. (1997). Practical digital libraries: books, bytes & bucks. San Francisco, CA: Morgan-Kaufmann Publishers.
- Lesk, M. E. (1999). Perspectives on DLI2 - growing the field. D-Lib Magazine, 5(7-8). Available at <http://www.dlib.org/dlib/july99/07lesk.html>
- Lutz, M. (1996). Programming python. Sebastopol CA: O'Reilly & Associates.
- Marazakis, M., Papadakis, D. & Papadakis, S. A. (1998). A framework for the encapsulation of value-added services in digital objects. In C. Nikolaou & C. Stephanidis (eds.) Research and advanced technology for digital libraries, second European conference, ECDL '98 (pp. 75-94). Berlin: Springer.
- Maly, K., French, J., Fox, E. & Selman, A. (1995). Wide area technical report service: technical reports online. Communications of the ACM, 38(4), 45.

- Maly, K., Nelson, M. L., & Zubair, M. (1999). Smart objects, dumb archives: a user-centric, layered digital library framework. *D-Lib Magazine*, 5(3). Available at <http://www.dlib.org/dlib/march99/maly/03maly.html>
- McGrath, R. E. (1996). Performance of several Web server platforms. National Center for Supercomputing Applications Technical Report. Available at <http://www.ncsa.uiuc.edu/InformationServers/Performance/Platforms/report.html>
- Miller, E. (1998). An introduction to the Resource Description Framework. *D-Lib Magazine*, 4(5). Available at <http://www.dlib.org/dlib/may98/miller/05miller.html>
- Monostori, K., Zaslavsky, A. & Schmidt, H. (2000). Document overlap detection systems for distributed digital libraries. Proceedings of the fifth ACM conference on digital libraries (pp. 226-227), San Antonio, TX.
- Mori, R. & Kawahara, M. (1990). Superdistribution: the concept and the architecture. *Transactions of the IEICE*, E73(7). Available at <http://www.virtualschool.edu/mon/ElectronicProperty/MoriSuperdist.html>
- NASA (1998). NASA Scientific and Technical Information (STI) program plan. Available at <http://stipo.larc.nasa.gov/splan/>
- Nebel, E. & Masinter, L. (1995). Form-based file upload in HTML. Internet RFC-1867. Available at <ftp://ftp.isi.edu/in-notes/rfc1867.txt>
- Nelson, C. (1995). OpenDoc and its architecture. *The X Resource*, 1(13), 107-126.
- Nelson, M. L. & Gottlich, G. L. (1994) Electronic document distribution: design of the anonymous FTP Langley technical report server, NASA-TM-4567, March 1994. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/tm4567.pdf>
- Nelson, M. L., Gottlich, G. L., & Bianco, D. J. (1994). World Wide Web implementation of the Langley technical report server. NASA TM-109162. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/tm109162.pdf>
- Nelson, M. L., Gottlich, G. L., Bianco, D. J., Paulson, S. S., Binkley, R. L., Kellogg, Y. D., Beaumont, C. J., Schmunk, R. B., Kurtz, M. J., Accomazzi, A., & Syed, O. (1995). The NASA technical report server. *Internet Research: Electronic Network Applications and Policy*, 5(2), 25-36. Available at <http://techreports.larc.nasa.gov/ltrs/papers/NASA-95-ir-p25/NASA-95-ir-p25.html>

- Nelson, M. L. & Esler, S. L. (1997). TRSkit: a simple digital library toolkit. *Journal of Internet Cataloging*, 1(2), 41-55. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/1997/jp/NASA-97-jic-mln.pdf>
- Nelson, M. L., Maly, K., Shen, S. N. T., & Zubair, M. (1998). NCSTRl+: adding multi-discipline and multi-genre support to the Dienst protocol using clusters and buckets. *Proceedings of the IEEE forum on research and technology advances in digital libraries* (pp. 128-136), Santa Barbara, CA. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/1998/mtg/NASA-98-ieeeedl-mln.pdf>
- Nelson, M. L. (1999). A digital library for the National Advisory Committee for Aeronautics. NASA/TM-1999-209127. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/1999/tm/NASA-99-tm209127.pdf>
- Nelson, M. L., Maly, K., Croom, D. R., & Robbins, S. W. (1999). Metadata and buckets in the smart object, dumb archive (SODA) Model, *Proceedings of the third IEEE meta-data conference*, Bethesda, MD. Available at <http://www.computer.org/proceedings/meta/1999/papers/53/mnelson.html>
- Ockerbloom, J. (1998). Mediating among diverse data formats. Ph.D. Dissertation, Carnegie Mellon University, CMU-CS-98-102. Available at <http://reports-archive.adm.cs.cmu.edu/anon/1998/abstracts/98-102.html>
- Odlyzko, A. M. (1995). Tragic loss or good riddance? The impending demise of traditional scholarly journals. *International Journal of Human-Computer Studies*, 42, 71-122.
- Olson, M. A., Bostic, K. & Seltzer, M. (1999). Berkeley DB. *Proceedings of the 1999 USENIX annual technical conference*, Monterey, CA.
- Ousterhout, J. K. (1994). Tcl and the Tk toolkit. Reading, MA: Addison-Wesley.
- Paepcke, A. (1996). Digital libraries: searching is not enough. *D-Lib Magazine* 2(5). Available at <http://www.dlib.org/dlib/may96/stanford/05paepcke.html>
- Paepcke, A. (1997). InterBib: bibliography-related services. Available at <http://www-diglib.stanford.edu/~testbed/interbib/>
- Paskin, N. (1999). DOI: current status and outlook. *D-Lib Magazine*, 5(5). Available at <http://www.dlib.org/dlib/may99/05paskin.html>

- Patterson, David A. (1994). How to have a bad career in research/academia. Keynote Address at the First Symposium on Operating System Design and Implementation, Monterey, CA. Available at <http://http.cs.berkeley.edu/~patterson/talks/bad.ps>
- Phelps, T. A. & Wilensky, R. (2000). Multivalent documents. *Communications of the ACM*, 43(6), 83-90.
- Powell, A. L. & French, J. C. (2000). Growth and server availability of the NCSTRL digital library. *Proceedings of the fifth ACM conference on digital libraries* (pp. 264-265), San Antonio, TX. Available at <http://www.cs.virginia.edu/~cyberia/papers/DL00.pdf>
- Rasmussen, E. (1992). Clustering algorithms. In W. B. Frakes & R. Baeza-Yates (Eds.), *Information retrieval: data structures & algorithms* (pp. 363-392), Upper Saddle River, NJ: Prentice-Hall.
- Rivest, R. (1992). The MD5 message-digest algorithm. Internet RFC-1321. Available at <ftp://ftp.isi.edu/in-notes/rfc1321.txt>
- Rocha, L. M. (1999). TalkMine and the adaptive recommendation project. *Proceedings of the fourth ACM conference on digital libraries* (pp. 242-243), Berkeley, CA. Available at <http://www.c3.lanl.gov/~rocha/dl99.html>
- Roper, D. G., McCaskill, M. K., Holland, S. D., Walsh, J. L., Nelson, M. L., Adkins, S. L., Ambur, M. Y., & Campbell, B. A. (1994). A strategy for electronic dissemination of NASA Langley publications. NASA TM-109172. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/tm109172.pdf>
- Rothenberg, J. (1995). Ensuring the longevity of digital documents. *Scientific American*, 272(1), 42-47.
- Salampsasis, M., Tait, J. & Hardy, C. (1996). An agent-based hypermedia framework for designing and developing digital libraries. *Proceedings of the third forum on research and technology advances in digital libraries* (pp. 5-13), Washington DC.
- Salton, G. & Lesk, M. E. (1968). Computer evaluation of indexing and text processing, *Journal of the Association of Computing Machinery*, 15(1), 8-36.
- Sanchez, J. A., Legget, J. J., & Schnase, J. L. (1997). AGS: introducing agents as services provided by digital libraries. *Proceedings of the second ACM international conference on digital libraries* (pp. 75-82), Philadelphia, PA.

- Sanchez, J. A., Lopez, C. A., & Schnase, J. L. (1998). An agent-based approach to the construction of floristic digital libraries. *Proceedings of the third ACM international conference on digital libraries* (pp. 210-216), Pittsburgh, PA.
- Schatz, B., & Chen, H. (1996). Building large-scale digital libraries. *IEEE Computer*, 29(5), 22-26.
- Scherlis, W. L. (1996). Repository interoperability workshop: towards a repository reference model. *D-Lib Magazine*, 2(10). Available at <http://www.dlib.org/october96/workshop/10scherlis.html>
- Scott, E. W. (1953). New patterns in scientific research and publication. *American Documentation*, 4(3), 90-95.
- Shafer, K., Weibel, S., Jul, E. & Fausey, J. (1996). Introduction to persistent uniform resource locators. *Proceedings of INET 96*, Montreal, Canada. Available at <http://purl.oclc.org/OCLC/PURL/INET96>
- Shivakumar, N. & Garcia-Molina, H. (1995). SCAM: a copy detection mechanism for digital documents. *Proceedings of the second international conference in theory and practice of digital libraries* (pp. 155-163), Austin, TX.
- Shklar, L., Makower, D., Maloney, E. & Gurevich (1998). An application development framework for the virtual Web. *Proceedings of the fourth international conference on information systems, analysis, and synthesis*, Orlando, FL. Available at <http://www.cs.rutgers.edu/~shklar/isas98/>
- Sibert, O., Bernstein, D. & Van Wie, D. (1995). DigiBox: a self-protecting container for information commerce. *Proceedings of the first USENIX workshop on electronic commerce*, New York, NY.
- Sobieski, J. (1994). A proposal: how to improve NASA-developed computer programs. *NASA CP-10159*, pp. 58-61.
- Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1), 11-20.
- Sparck Jones, K. (1979). Experiments in relevance weighting of search terms. *Information Processing and Management*, 15(3), 133-144.
- Stern, I. (1995). Scientific data format information FAQ. Available at <http://www.faqs.org/faqs/sci-data-formats/>

- Stein, L. (1998). Official guide to programming with CGI.pm. New York, NY: John Wiley & Sons.
- Stein, L., MacEachern, D. & Mui, L. (1999). Writing Apache modules in Perl and C: the Apache API and mod_perl. Sebastopol CA: O'Reilly & Associates.
- Steiner, J. G., Neuman, C. & Schiller, J. I. (1988). Kerberos: an authentication service for open network systems. Proceedings of the winter 1988 USENIX conference (pp. 191-202), Dallas, TX.
- Sullivan, W. T. III, Werthimer, D., Bowyer, S., Cobb, J., Gedye, D. & Anderson, D. (1997). A new major SETI project based on Project Serendip data and 100,000 personal computers. In C.B. Cosmovici, S. Bowyer, & D. Werthimer (Eds.), Astronomical and biochemical origins and the search for life in the universe, Bologna, Italy: Editrice Compositori. Available at http://setiathome.berkeley.edu/woody_paper.html
- Sun Microsystems, Inc. (1999). The maximum number of directories allowed on Solaris is limited by the LINK_MAX parameter. InfoDoc # 19895.
- Sun, S. X. & Lannom, L. (2000). Handle system overview. Internet Draft. Available at <http://www.ietf.org/internet-drafts/draft-sun-handle-system-04.txt>
- Task Force on Archiving of Digital Information (1996). Preserving digital information. Available at <http://www.rlg.org/ArchTF/>
- Tiffany, M. E. & Nelson, M. L. (1998). Creating a canonical scientific and technical information classification system for NCSTRL+. NASA/TM-1998-208955. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/1998/tm/NASA-98-tm208955.pdf>
- United States General Accounting Office (1990). NASA is not properly safeguarding valuable data from past missions, GAO/IMTEC-90-1.
- Van de Sompel, H. & Hochstenbach, P. (1999). Reference linking in a hybrid library environment: part 2: SFX, a generic linking service. D-Lib Magazine 5(4). Available at http://www.dlib.org/dlib/april99/van_de_sompel/04/van_de_sompel-pt2.html

- Van de Sompel, H., Krichel, T., Nelson, M. L., Hochstenbach, P., Lyapunov, V. M., Maly, K., Zubair, M., Kholief, M., Liu, X. & O'Connell, H. (2000a). The UPS prototype: an experimental end-user service across e-print archives. *D-Lib Magazine*, 6(2). Available at <http://www.dlib.org/dlib/february00/vandesompel-ups/02vandesompel-ups.html>
- Van de Sompel, H., Krichel, T., Nelson, M. L., Hochstenbach, P., Lyapunov, V. M., Maly, K., Zubair, M., Kholief, M., Liu, X. & O'Connell, H. (2000b). The UPS prototype project: exploring the obstacles in creating across e-print archive end-user service, Old Dominion University Computer Science Technical Report TR 2000-01. Available at http://ncstrl.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.odu_cs/TR_2000_01
- Van de Sompel, H. & Lagoze, C. (2000). The Santa Fe Convention of the Open Archives initiative. *D-Lib Magazine*, 6(2). Available at <http://www.dlib.org/dlib/february00/vandesompel-oai/02vandesompel-oai.html>
- Vickery, B. (1999). A century of scientific and technical information. *Journal of Documentation*, 55(5), 476-527.
- Vinoski, S. (1997). CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 4(2), 46-55.
- Wall, L., Christiansen, T. & Schwartz, R. L. (1996). *Programming Perl*. Sebastopol, CA: O'Reilly & Associates, Inc.
- Waugh, A., Wilkinson, R., Hills, B., & Dellóro, J. (2000). Preserving digital information forever. *Proceedings of the fifth ACM conference on digital libraries* (pp. 175-184), San Antonio, TX.
- Weibel, S., Kunze, J., Lagoze, C. & Wolfe, M. (1998). Dublin Core metadata for resource discovery. Internet RFC-2413. Available at <ftp://ftp.isi.edu/in-notes/rfc2413.txt>
- Yeong, W., Howes, T. & Kille, S. (1995). Lightweight directory access protocol. Internet RFC-1777. Available at <ftp://ftp.isi.edu/in-notes/rfc1777.txt>

APPENDIX A

BUCKET VERSION HISTORY

Version	Date	Kilobytes	Inodes	Comments
“proto buckets” of the NACATRS	January 1996	n/a	n/a	Not really a bucket, but the concept for buckets grew out of the experiences from this project.
version 0	July 1997	n/a	n/a	First digital object to be identified as a bucket. Used only for research purposes: refining the bucket concept & defining the API. Structural design is completely different.
version 1.0	July 1998	n/a	n/a	complete re-write of version 0; the design of the current buckets traces to this version.
version 1.1	August 1998	n/a	n/a	First implementation of the current T&C design.
version 1.11	September 1998	n/a	n/a	Significant change in parsing of metadata. Name collisions handled.
version 1.12	September 1998	n/a	n/a	T&C changes.
version 1.13	October 1998	n/a	n/a	Fixed problems with self-deleting buckets in version 1.12

version 1.2	November 1998	97	40	The first public release of buckets. Has only a basic set of methods and simple T&C support. Display of metadata is improved. Bucket more tolerant of variations in internal structure.
version 1.3	July 1999	118	53	Method set expanding to influence appearance and behavior of bucket. Packages are locked out from http browsing (true data hiding).
version 1.3.1	July 1999	125	58	Can now distribute different types of metadata if they have been pre-loaded. More appearance/behavior methods evolving.
version 1.3.2	July 1999	125	58	Minor bug-fix.
UPS version 1.6 (based on version 1.3.2)	October 1999	97	56	Final version of the template used in the UPS project. Based on the 1.3.2 template, the UPS template was slightly optimized for storage efficiency, and introduced some of the new functionality in later bucket versions.
version 1.4	December 1999	134	62	Code factoring now possible. Many of the appearance and

				behavior models have been collapsed into preferences. “display” method borrows heavily from UPS look and feel.
version 1.5	February 1999	145	68	“pack” and “unpack” methods implemented to assist with bucket mobility. “display” method can take several arguments for customizing its output.
version 1.5.1	February 1999	148	70	Group T&C support for IP addresses and hostnames added.
version 1.5.2	February 1999	149	70	Minor bug fix.
version 1.5.3	March 1999	148	70	Minor bug fixes.
version 1.5.4	March 1999	147	70	More minor bug fixes.
version 1.5.5	April 1999	143	66	Naming of metadata changed for the “display” method to be inline with that used in the NCSTRL+ project.
version 1.6	April 1999	144	68	Buckets now BCS aware, especially with respect to metadata conversion. Buckets can now send email when events occur. Many bug fixes and optimizations.

Prior to version 1.2, source code releases were not preserved. Source code and detailed release notes versions after 1.2 can be found at:

<http://dlib.cs.odu.edu/bucket-shop/>

APPENDIX B

BUCKET API

Method:	add_element
Arguments:	element_name, pkg_name, element_bib, upfile
Returned MIME Type:	text/plain
Example(s):	?method=add_element&element_name=bar.pdf& pkg_name=foo.pkg&element_bib=X&upfile=X
Discussion:	“add_element” adds a specified element to a specified package in the bucket. “element_name” is the name the element will be stored as in the bucket, and “pkg_name” is the name of the package that the element will be put into. “element_bib” contains the RFC-1807 description of the element, and “upfile” contains the actual file, uploaded as described in RFC-1867 (Nebel & Masinter, 1995).
See Also:	delete_element, display

Method:	<code>add_method</code>
Arguments:	<code>target</code> , <code>upfile</code>
Returned MIME Type:	<code>text/plain</code>
Example(s):	<code>?method=add_method&target=foo&upfile=X</code>
Discussion:	<p>“<code>add_method</code>” adds a method to a bucket by uploading Perl source code into the bucket. “<code>target</code>” specifies the name of the new method, while “<code>upfile</code>” contains the source code uploaded as described in RFC-1867. The bucket performs no error checking on the uploaded source code; if the user can satisfy the T&C for “<code>add_method</code>”, it is assumed they know what they are doing.</p>
See Also:	<code>delete_method</code> , <code>list_methods</code>

Method:	add_package
Arguments:	pkg_name, pkg_bib
Returned MIME Type:	text/plain
Example(s):	?method=add_package&pkg_name=foo.pkg&pkg_bib=X
Discussion:	<p>“add_package” creates a new package in the bucket. On a subsequent “display” of the bucket, the new package will appear at the end of the list of previous packages.</p> <p>“pkg_name” is the name of the package to be added. If the package name does not include an extender of “.pkg”, one will be added. “pkg_bib” contains the RFC-1807 description of the package.</p>
See Also:	delete_package

Method:	add_principal
Arguments:	principal, passwd, epasswd
Returned MIME Type:	text/plain
Example(s):	?method=add_principal&principal=bob&passwd=secret ?method=add_principal&principal=bob&epasswd=4Rals3Q
Discussion:	“add_principal” adds a user with a password to the bucket’s internal list of recognized principals that can be named in its T&C files. “principal” is the name of the user to be defined. “passwd” is the clear text version of the password, and “epasswd” is the encrypted (with Unix crypt(3)) version of the password. Only one of the two password arguments needs to be supplied.
See Also:	delete_principal, list_principal

Method:	add_tc
Arguments:	target, value
Returned MIME Type:	text/plain
Example(s):	?method=add_tc&target=delete_bucket&value=X
Discussion:	<p>“add_tc” uploads (as described in RFC-1867) a T&C file that defines who can invoke the method named by ‘target’.</p> <p>“value” contains the actual file contents; the syntax of which is described in chapter three.</p>
See Also:	delete_tc, list_tc

Method:	delete_bucket
Arguments:	none
Returned MIME Type:	text/plain
Example(s):	?method=delete_bucket
Discussion:	“delete_bucket” deletes the entire bucket – no confirmation is requested. This is a very dangerous method, and because of this it is disabled in the standard bucket distribution.
See Also:	delete_element, delete_package

Method:	<code>delete_element</code>
Arguments:	<code>element_name</code> , <code>pkg_name</code>
Returned MIME Type:	<code>text/plain</code>
Example(s):	<code>?method=delete_element&element_name=bar.pdf&pkg_name=foo.pkg</code>
Discussion:	“ <code>delete_element</code> ” deletes the element named by “ <code>element_name</code> ” and “ <code>pkg_name</code> ”. It will also delete the RFC-1807 metadata associated with the named element.
See Also:	<code>add_element</code> , <code>delete_package</code>

Method:	delete_log
Arguments:	log
Returned MIME Type:	text/plain
Example(s):	?method=delete_log&log=access.log
Discussion:	<p>“delete_log” deletes the log named by the argument “log”. This method is intended to be used by tools or cron jobs to autoally harvest and then prune bucket logs, which would otherwise grow without bound.</p>
See Also:	get_log, list_logs

Method:	delete_method
Arguments:	target
Returned MIME Type:	text/plain
Example(s):	?method=delete_method&target=add_principal
Discussion:	“delete_method” deletes the method named in “target” from the bucket. Calling “delete_method” with arguments of “add_method” then “delete_method” would insure a static bucket whose methods could not be changed through the API.
See Also:	add_method, list_methods

Method:	delete_package
Arguments:	pkg_name
Returned MIME Type:	text/plain
Example(s):	?method=delete_package&pkg_name=foo.pkg
Discussion:	<p>“delete_package” deletes the entire package named in “pkg_name”. Any elements residing in the package will be deleted at the same time; no confirmation will be requested. The relevant metadata fields will be deleted as well.</p>
See Also:	add_package

Method:	delete_principal
Arguments:	principal
Returned MIME Type:	text/plain
Example(s):	?method=delete_principal&principal=bob
Discussion:	“delete_principal” will delete the user named by the “principal” argument from the bucket’s internal list of named principals. The user’s password will also be deleted.
See Also:	add_principal, list_principals

Method:	delete_tc
Arguments:	target
Returned MIME Type:	text/plain
Example(s):	?method=delete_tc&target=display.tc
Discussion:	“delete_tc” deletes the T&C file named by “target”. Note that “edit_tc” functionality would be accomplished by a “list_tc” / “delete_tc” / “add_tc” series of calls.
See Also:	add_tc, list_tc

Method:	display
Arguments:	none, bold, view, sfx, redirect, pkg_name, element_name, page, thumbnail
Returned MIME Type:	text/html; other MIME types as appropriate
Example(s):	<p>?method=display</p> <p>?method=display&bold=aircraft+engine</p> <p>?method=display&view=staff</p> <p>?method=display&sfx=http://sfx.foo.edu/</p> <p>?method=display&redirect=http://www.foo.edu/</p> <p>?method=display&pkg_name=foo.pkg&element_name=bar.pdf</p> <p>?method=display&pkg_name=foo.pkg&element_name=bar.scan&page=0001.gif</p> <p>?method=display&pkg_name=foo.pkg&element_name=bar.scan&thumbnail=1</p>
Discussion:	<p>“display” is easily the most complex bucket method. When called with no arguments, it generates an HTML human-readable listing of the bucket contents. “bold”, “view”, and “sfx” can all be used to describe the normal HTML bucket listing. “bold” takes a list of keywords that displays them in bold during the bucket display. “view” defines an alternate display criteria for the bucket, which can be used to implement role based displays. “sfx” provides the location of a SFX server. “redirect” causes the bucket to generate an http 302 response and redirect the browser to the specified URL. When “pkg_name” and “element_name” are specified, the bucket returns the named element, and gives it a MIME</p>

type based on the bucket's own internal listing of file extenders and MIME types. If "element_name" ends in ".scan", then either "page" or "thumbnail" can be specified. "thumbnail" will generate a listing of N thumbnail GIFs which correspond to scanned pages of a document, and where N specified by the preference "thumbnail_increment". "page" shows only the large GIF of a scanned page.

See Also:

none

Method:	get_log
Arguments:	log
Returned MIME Type:	text/plain
Example(s):	?method=get_log&log=access.log
Discussion:	“get_log” returns the entire log specified in the “log” argument. Currently, buckets only maintain a single log by default, but this could change in future versions. Also, local implementations of buckets are free to implement their own logs.
See Also:	delete_log

Method:	get_preference
Arguments:	none, pref
Returned MIME Type:	text/plain
Example(s):	?method=get_preference ?method=get_preference&pref=logging
Discussion:	“get_preference” with no arguments returns the current values of all the buckets defined preferences. If a single preference is specified in the argument “pref”, then only its value is displayed.
See Also:	set_preference

Method:	<code>get_state</code>
Arguments:	<code>state</code>
Returned MIME Type:	<code>text/plain</code> ; or other as appropriate
Example(s):	<code>?method=get_state&state=approved</code>
Discussion:	“ <code>get_state</code> ” returns the value of the bucket state variable specified in the argument “ <code>state</code> ”. Buckets do not internally use these state variables – they are for use by external tools that wish to leave the bucket in a certain state. State variables are of type <code>text/plain</code> , but they can be any MIME type.
See Also:	<code>set_state</code>

Method:	id
Arguments:	none
Returned MIME Type:	text/plain
Example(s):	?method=id
Discussion:	“id” returns the id for the bucket as specified in the RFC-1807 “ID::” metadata line.
See Also:	none

Method:	lint
Arguments:	none
Returned MIME Type:	text/plain
Example(s):	?method=lint
Discussion:	<p>“lint” performs a series of internal checks on the bucket. These include: comparing the packages and elements listed in the metadata to those physically in the bucket; verifying that all packages are closed to http browsing; verifying that files are writable by the http server; and comparing the URL used to access the bucket with that expect in the metadata.</p>
See Also:	none

Method:	list_logs
Arguments:	none
Returned MIME Type:	text/plain
Example(s):	?method=list_logs
Discussion:	<p>“list_logs” returns a list of all the logs defined for the bucket. Currently, buckets only maintain a single log by default, but this could change in future versions. Also, local implementations of buckets are free to implement their own logs.</p>
See Also:	delete_log, get_log

Method:	list_methods
Arguments:	none
Returned MIME Type:	text/plain
Example(s):	?method=list_methods
Discussion:	“list_methods” returns a list of all the methods defined for the bucket. This list is expected to be different for locally modified buckets, which could add or delete methods from the default set.
See Also:	add_method, delete_method

Method:	list_principals
Arguments:	none
Returned MIME Type:	text/plain
Example(s):	?method=list_principals
Discussion:	“list_principals” lists all the defined users for the bucket. Passwords are obviously not included in the display.
See Also:	add_principal, delete_principal

Method:	<code>list_source</code>
Arguments:	<code>none</code> , <code>target</code>
Returned MIME Type:	<code>text/plain</code>
Example(s):	<code>?method=list_source</code> <code>?method=list_source&target=display</code>
Discussion:	<p>“<code>list_source</code>” returns the Perl source code used by the bucket. If no arguments are given, the source code for the “<code>index.cgi</code>” is returned. Otherwise, the source code for the method specified in the argument “<code>target</code>” is returned.</p>
See Also:	<code>add_method</code> , <code>delete_method</code> , <code>list_methods</code>

Method:	list_tc
Arguments:	none, target
Returned MIME Type:	text/plain
Example(s):	?method=list_tc ?method=list_tc&target=display
Discussion:	“list_tc” with no arguments lists all the methods for the bucket that have T&C files and lists the file contents. If no T&C are defined, nothing is returned. If the argument “target” is supplied, “list_tc” will return only the T&C file for the method specified by “target”, or nothing if no T&C are defined for that method.
See Also:	add_tc, delete_tc

Method:	metadata
Arguments:	none, format
Returned MIME Type:	text/plain (or text/xml as appropriate)
Example(s):	?method=metadata ?method=metadata&format=oams
Discussion:	“metadata” invoked with no arguments returns the metadata in RFC-1807 format. If you specify a different metadata format in the “format” argument, it will first look to see if thatrmat is stored internally in the bucket, and if so determine if it is clean. If the bucket does not have the format, or it is dirty, it will contact the BCS and attempt to convert the RFC-1807 format to the requested format, if the BCS can do that conversion. If the BCS cannot convert to that format, an error is returned.
See Also:	set_metadata

Method:	pack
Arguments:	none, name, type, format
Returned MIME Type:	application/tar (or other MIME types as appropriate)
Example(s):	?method=pack ?method=pack&name=foo.pkg ?method=pack&type=payload ?method=pack&format=tar
Discussion:	<p>“pack” takes a number of interchangeable arguments, but all have default values so “pack” can be invoked sans arguments. “type” specifies one of: bucket (entire bucket, default value), package (package name specified in “name”), payload (user portion of the bucket only), or ride (internal structure of the bucket only). “format” currently only recognizes the value “tar”, but this should change in the future (though “tar” will remain the default value). “pack” will generate a stream of either the entire bucket or just the requested part of a bucket. This stream can be used to overwrite an existing bucket, similar to the Unix fork()/exec() model.</p>
See Also:	unpack

Method:	set_metadata
Arguments:	name, upfile
Returned MIME Type:	text/plain
Example(s):	?method=set_metadata&name=metadata.oams&upfile=X
Discussion:	“set_metadata” writes the metadata file named in “name” and supplied in “upfile” to the bucket. This method is useful in either overwriting the RFC-1807 metadata (perhaps to correct errors), or to upload other metadata formats; either from BCS or entirely different formats that the BCS does not know about.
See Also:	metadata

Method:	set_preference
Arguments:	pref, upfile
Returned MIME Type:	text/plain
Example(s):	?method=set_preference&pref=framable&upfile=no
Discussion:	“set_preference” writes the preference named in “pref” and takes the value specified in “upfile”.
See Also:	get_preference

Method:	set_state
Arguments:	state, upfile
Returned MIME Type:	text/plain
Example(s):	?method=set_state&state=approved&upfile=yes
Discussion:	“set_state” writes the state variable named in “state” and takes the value specified in “upfile”. “upfile” does not have to be of type text/plain.
See Also:	get_state

Method:	set_version
Arguments:	value
Returned MIME Type:	text/plain
Example(s):	?method=set_version&value=2.0
Discussion:	“set_version” sets the version of the bucket to the text string specified in “value”.
See Also:	version

Method:	unpack
Arguments:	format, type, upfile
Returned MIME Type:	text/plain
Example(s):	?method=unpack&upfile=X ?method=unpack&format=tar&upfile=X ?method=unpack&format=tar&type=bucket&upfile=X
Discussion:	“unpack” takes the bucket stream specified in “upfile” and puts it into the bucket. “format” can be specified, but the only currently defined format is “tar”. “type” specifies one of: bucket (entire bucket, default value), package (package name specified in “name”), payload (user portion of the bucket only), or ride (internal structure of the bucket only).
See Also:	pack

Method:	version
Arguments:	none
Returned MIME Type:	text/plain
Example(s):	?method=version
Discussion:	“version” returns a text string to describe what type of bucket it is. There is no structure imposed on what this string can be.
See Also:	set_version

APPENDIX C

DA API

Method:	da_delete
Arguments:	id
Returned MIME Type:	text/plain
Example(s):	?method=da_delete&id=1234
Discussion:	“da_delete” removes the object specified by the argument “id” from the archive. “id” has no built in assumptions regarding what type of unique identifier is used: CNRI handles, DOIs, URLs, etc. all could be used. Currently, “da_delete” does not return an error if “id” is not present in the archive.
See Also:	da_put, da_list

Method:	da_get
Arguments:	id, url
Returned MIME Type:	text/plain
Example(s):	?method=da_get&url=http://foo.edu/1234/ ?method=da_get&id=1234
Discussion:	<p>“da_get” is an optional method for the DA; it is used primarily to build a model where the archive still contains some control over the access of the bucket, so that rather than going directly to the bucket, the archived is asked to redirect the user to the bucket. The buckets could be modified to only accept responses originating from an archive, and the DA could have regular bucket T&C controlling the behavior of “da_get”. Either argument “id” or “url” can be specified, and “url” has precedence over “id” if both are specified. The archive issues an http status code 302 for URL redirection.</p>
See Also:	none

Method:	da_info
Arguments:	none
Returned MIME Type:	text/plain
Example(s):	?method=da_info
Discussion:	<p>“da_info” currently takes no arguments and simply returns the element stored in <code>holdings.pkg/info.txt</code>. The purpose of this method is to return a machine readable description of the archive, its capabilities and its holdings. Human readable descriptions would be available through the standard bucket method “display”. “da_info” should be expanded to take arguments as to which format it would like the archive information (and the MIME type set accordingly), with possible conversion from the BCS.</p>
See Also:	none

Method:	da_list
Arguments:	id, url, adate, pdate, subject, metadata
Returned MIME Type:	text/plain
Example(s):	<p>?method=da_list</p> <p>?method=da_list&metadata=on</p> <p>?method=da_list&id=1234</p> <p>?method=da_list&url=http://foo.edu/1234/</p> <p>?method=da_list&adate=19990101</p> <p>?method=da_list&pdate=19930430</p> <p>?method=da_list&pdate=>19931231</p> <p>?method=da_list&adate=19891231-20000101</p> <p>?method=da_list&adate=subject=computer_science</p>
Discussion:	<p>“da_list” is the primary method of the DA. This is the method that will be used by digital library services to determine what contents an archive has, what contents have changed since a specific date, and so forth. There are many arguments, all of which can be combined in various forms to select which ids (and URLs) will be returned. If no argument is specified, all the archive’s contents are returned. If a specific “id” or “url” is requested, then “da_list” is used for an existence test, yielding the id/url if the object exists, and nothing if it does not. There are three pre-defined “clusters” (in NCSTRL+ terminology) defined for the DA: “adatae” (accession date), “pdate” (publication date), “subject”. Both date fields follow the format of YYYYMMDD. All dates are non-inclusive. Dates can be specific days, or modified with “<”, “>”, or “-“ for less than, greater than, and range,</p>

respectively. “<” and “>” must precede the date, and “-“ must have a valid date on either side. The date modifiers cannot be combined. Regular expressions in the dates (i.e. “1999.*” for the entire calendar year 1999) are not supported. “subject” can be any text string from any subject classification system; there are no syntactic restrictions on “subject”. If “metadata” is set to any value, the metadata for the object(s) (if uploaded) will be returned as well.

See Also:

da_delete, da_put

Method:	da_put
Arguments:	id, url, adate, pdate, subject, metadata
Returned MIME Type:	text/plain
Example(s):	?method=da_put&id=1234&url=http://foo.edu/1234/&adate=19990215&pdate=19580413&subject=aeronautics&metadata=X
Discussion:	<p>“da_put” places an object in the DA. There are many arguments, but only “id” is mandatory – the others are optional. “id” can be a unique identifier in any format, “url” is a regular URL, “adate” (accession date) and “pdate” (publication date) are date strings in the format YYYYMMDD, and “subject” can be from any subject/discipline nomenclature. “metadata” is the objects metadata uploaded as per RFC-1867.</p>
See Also:	da_delete, da_list

APPENDIX D

BCS API

Method:	bcs_convert_image
Arguments:	in_format, out_format, upfile
Returned MIME Type:	text/plain
Example(s):	?method=bcs_convert_image&in_format=ps&out_format=pdf&upfile=X
Discussion:	<p>“bcs_convert_image” takes the file uploaded as per RFC-1867 in the argument “upfile” and converts it to the type specified in the argument “out_format”. “upfile” is assumed to be of the type specified in “in_format” – no checking is done to verify that “upfile” is of type “in_format”. “in_format” and “out_format” currently have the following types defined (alternate values in parentheses):</p> <ul style="list-style-type: none"> - gif - jpeg (jpg) - tiff (tif) - png - ps - pdf
See Also:	bcs_convert_metadata

Method:	<code>bcs_convert_metadata</code>
Arguments:	<code>in_format</code> , <code>out_format</code> , <code>in_file</code>
Returned MIME Type:	<code>text/plain</code> or <code>text/xml</code>
Example(s):	<code>?method=bcs_convert_metadata&in_format=rfc1807&out_format=oams&in_file=X</code>
Discussion:	<p>“<code>bcs_convert_metadata</code>” takes the metadata file uploaded as per RFC-1867 in “<code>in_file</code>” and returns it converted to the metadata format specified in “<code>out_format</code>”. The format of “<code>in_file</code>” is specified in “<code>upfile</code>”, but no checking is done to verify the accuracy between “<code>in_file</code>” and “<code>in_format</code>”.</p> <p>“<code>in_format</code>” has the following values defined:</p> <ul style="list-style-type: none"> - <code>refer</code> - <code>dublincore</code> - <code>rfc1807</code> - <code>bibtex</code> <p>“<code>out_format</code>” has the following values defined:</p> <ul style="list-style-type: none"> - <code>refer</code> - <code>dublincore</code> - <code>rfc1807</code> - <code>bibtex</code> - <code>oams</code>
See Also:	<code>bcs_convert_image</code>

Method:	bcs_list
Arguments:	id, url
Returned MIME Type:	text/plain
Example(s):	?method=bcs_list ?method=bcs_list&url=http://foo.edu/1234/ ?method=bcs_list&id=1234
Discussion:	“bcs_list” with no arguments lists all the ids (and URLs) of all the buckets that are registered with that BCS server. If either of “url” or “id” is specified, “bcs_list” acts as a test for existence; if the bucket identified by either “url” or “id” is registered with this BCS server, “bcs_list” will return its id or URL and will return nothing if the bucket is not registered.
See Also:	bcs_register, bcs_unregister

Method:	bcs_match
Arguments:	threshold, link, report, ids
Returned MIME Type:	text/plain
Example(s):	<p>?method= bcs_match</p> <p>?method=bcs_match&threshold=0.70</p> <p>?method= bcs_match&link=on</p> <p>?method= bcs_match&report=on</p> <p>?method= bcs_match=ids=1234+2345+1122</p>
Discussion:	<p>“bcs_match” performs similarity matching on registered buckets using the cosine correlation with frequency term-weighting measure. “bcs_match” can be a computational expensive task and run for a long time (see Chapter Five for details). Because the run time of a “bcs_match” session can be much longer than an average WWW browser session, “bcs_match” forks off a copy of itself to run on the server so that it cannot be killed from the browser. There are a number of arguments to “bcs_match”, all of which can be combined with each other. “threshold” defines a number between 0-1 for determining what constitutes “similar” documents (the default value is 0.85). If “link” is set to any value, “bcs_match” will upon completion of the similarity matching attempt to automatically create the linkages between the similar buckets. It will attempt to create a <code>BCS_Similarity.pkg</code> package if one does not exist, and then add the similar bucket, if not linked already (default action is not to link). If “report” is set to any value, “bcs_match” will record the output of this run in the element</p>

`matching` in the package `bcs.pkg` (default action is not to record the output). If “ids” has 1 or more values, then “bcs_match” will only compare the specified ids against the entire list of registered buckets (the default action is to compare all buckets against all buckets).

See Also:

none

Method:	bcs_message
Arguments:	search, replace, mesg, repeat
Returned MIME Type:	text/plain
Example(s):	<p>?method=bcs_message&search=NASA+Lewis</p> <p>?method=bcs_message&search=NASA+Lewis&replace=NASA+Glenn</p> <p>?method=bcs_message&search=foo&mesg=destroy_bucket</p> <p>?method=bcs_message&search=foo&mesg=destroy_bucket&repeat=1</p>
Discussion:	<p>“bcs_message” identifies buckets for communication purposes. “search” is a mandatory argument that specifies the regular expression to search for in all registered buckets. If no further arguments are given, “bcs_message” returns the ids or URLs of the buckets that have that regular expression. If “replace” is specified, the “search” regular expression is overwritten both in the registry and the bucket with the regular expression in “replace”. If “message” is specified, “bcs_message” sends all matching buckets the bucket message specified in “message”. If “repeat” is given an integer value, the “replace” or “mesg” actions are repeated up to (possibly less, depending on “search”) “repeat” times. Users should be aware when using “repeat” that no assumptions can be made on the order of how “bcs_message” finds the regular expression specified in “search”.</p>
See Also:	none

Method:	bcs_register
Arguments:	id, url, metadata
Returned MIME Type:	text/plain
Example(s):	?method=bcs_register&id=1234 ?method=bcs_register&id=1234&url=http://foo.edu/1234/ ?method=bcs_register&id=1234&metadata=X
Discussion:	“bcs_register” places the bucket specified by “id” or “url” in the bucket communication space. “id” is mandatory, but “url” is optional. A bucket cannot be the subject of “bcs_message”, “bcs_match” and other BCS methods until it has been registered. Although optional, “metadata” contains the metadata, uploaded as per RFC-1867, that will be used in “bcs_message” and “bcs_match”.
See Also:	bcs_list, bcs_unregister

Method:	bcs_unregister
Arguments:	id
Returned MIME Type:	text/plain
Example(s):	?method= bcs_unregister&id=1234
Discussion:	“bcs_unregister” removes the bucket specified by “id” from the bucket communication space. The corresponding URLs and metadata associated with that id will also be removed.
See Also:	bcs_list, bcs_register

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE August 2001		3. REPORT TYPE AND DATES COVERED Technical Memorandum
4. TITLE AND SUBTITLE Buckets: Smart Objects for Digital Libraries			5. FUNDING NUMBERS WU 992-16-05	
6. AUTHOR(S) Michael L. Nelson				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199			8. PERFORMING ORGANIZATION REPORT NUMBER L-18106	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001			10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA/TM-2001-211049	
11. SUPPLEMENTARY NOTES Also published as a PhD dissertation for the Old Dominion University, Computer Science Department, Norfolk, Virginia, August 2000.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified-Unlimited Subject Category 82 Distribution: Standard Availability: NASA CASI (301) 621-0390			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Current discussion of digital libraries (DLs) is often dominated by the merits of the respective storage, search and retrieval functionality of archives, repositories, search engines, search interfaces and database systems. While these technologies are necessary for information management, the information content is more important than the systems used for its storage and retrieval. Digital information should have the same long-term survivability prospects as traditional hardcopy information and should be protected to the extent possible from evolving search engine technologies and vendor vagaries in database management systems. Information content and information retrieval systems should progress on independent paths and make limited assumptions about the status or capabilities of the other. Digital information can achieve independence from archives and DL systems through the use of buckets. Buckets are an aggregative, intelligent construct for publishing in DLs. Buckets allow the decoupling of information content from information storage and retrieval. Buckets exist within the Smart Objects and Dumb Archives model for DLs in that many of the functionalities and responsibilities traditionally associated with archives are "pushed down" (making the archives "dumber") into the buckets (making them "smarter"). Some of the responsibilities imbued to buckets are the enforcement of their terms and conditions, and maintenance and display of their contents.				
14. SUBJECT TERMS WWW, Digital Libraries, Information Retrieval, STI, Archives			15. NUMBER OF PAGES 177	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	